Math

UIUCDCS-R-77-885

UILU-ENG 77 1742

NOR NETWORK TRANSDUCTION SYSTEM
(Principles of the NETTRA System)

by

K. C. Hu
S. Muroga

August 1977

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-77-885

NOR NETWORK TRANSDUCTION SYSTEM
(Principles of the NETTRA System)

by

K. C. Hu
S. Muroga

August, 1977

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

This work was supported in part by the National Science Foundation under Grant NSF DCR73-03421.

ABSTRACT

The network transduction programs, including NETTRA-PG1, -P1, -P2, -G1, -G2, -G3, -G4, -E1, and -E2 are combined as a large program system named the NETTRA system (NETwork TRAnsduction system).

The NETTRA system can design near-optimal, multiple-output, multi-level and loop-free NOR(NAND) networks under fan-in/fan-out restrictions and/or level restriction.

Given function(s) may be completely or incompletely specified and both complemented and uncomplemented external variables are permitted as inputs.

The user can specify the control sequence (the types of the initial network methods and the types and the order of the transduction procedures to be applied) to solve his problem. Besides, four control sequences are provided for the users who are not interested in the details of how to specify the control sequence.

Facilities for treating unfinished jobs due to the expiration of computation time are also provided by the system.

ACKNOWLEDGMENT

During the period of the research of NOR(NAND) network transduction methods, many people, besides the authors, were involved. They are Messrs. Y. Kambayashi, H. C. Lai, J. N. Culliney, K. Hohulin, B. Plangsiri, J. G. Legge, and R. Cutler.

The authors wish to express their thanks to Messrs. H. C. Lai and J. N. Culliney for valuable discussions and comments in designing the NETTRA system.

The excellent typing job done by Mrs. R. Taylor and Mrs. J. Wingler is also acknowledged.

TABLE OF CONTENTS

Page

1. INTRODUCTION

In the synthesis problem of logic networks, how to get derivation of optimal NOR(NAND) networks for a given switching function (or a set of switching functions) is important and interesting since basic logic gates of many integrated circuits (for example, RTL (Resistor-Transistor Logic), DTL (Diode-Transistor Logic), ECL (Emitter-Coupled Logic), IIL (Integrated-Injection Logic) and VMOS) realize NOR or NAND functions. Existing logic design methods for synthesizing NOR (NAND) networks are classified into the following five groups:

(1) Switching algebraic methods [7],[10],[27].

(2) Map factoring method [28],[31],[41].

(3) Exhaustive method [8].

(4) Integer programming methods (implicit enumeration method formulated inequalities [29],[30],[32],[33],[34] and branch-and-bound method without inequalities* [4],[37].)

(5) Transformation methods [5],[6],[16],[23],[28],[35].

The switching algebraic methods can produce optimal networks only under very specific constraints (such as two-level or three-level networks without fan-in, fan-out restrictions). The map factoring method is easy

*This branch-and-bound method finds optimal networks by using the concepts of covered and uncovered components, possible covers, selection criterion of uncovered components and desirability. It does not try to solve inequalities (like the integer programming method does).

to use by hand for problems with four or fewer external variables, but the optimality of the results is not guaranteed. The exhaustive method can be applied only to networks with very few gates (usually at most 5). As the number of gates in a network increases, it becomes excessively time-consuming to exhaust all feasible networks (networks which realize given functions and satisfy given restrictions) for the given problem. The integer programming methods produce optimal networks in much shorter time than the exhaustive method. (The integer programming methods consist of the implicit enumeration method based on inequality formulation and the branch-and-bound method without inequalities. The branch-and-bound method is usually more efficient than the implicit enumeration method, though it is much harder to implement [36].) But as the number of gates in a network exceeds 10, the integer programming methods (both the implicit enumeration method and the branch-and-bound method) become too time-consuming for practical use.

When people need a synthesis procedure which can treat a logic network with many gates, the network transformation methods seem promising although they do not guarantee the optimality of the results. Some NOR(NAND) network transformation methods were discussed in [5],[6],[28]. Recently more general transformations were studied by T. Nakagawa, H. C. Lai and S. Muroga (the transformations are combined with the branch-and-bound method in order to improve the efficiency of the branch-and-bound algorithm [35]), H. Lee and E. S. Davidson [23] and Y. Kambayashi, H. C. Lai and S. Muroga [16].

In the past few years, the research group of S. Muroga developed a new approach for the design of near-optimal NOR(NAND) networks. This new approach is named the transduction methods (transformation and reduction). Any network designed by some known method, which is called an initial network, is transformed and reduced into a simpler network by the transduction methods.*

The transduction methods are based on the concepts of permissible functions which were originated by Y. Kambayashi and S. Muroga [17]. Y. Kambayashi, H. C. Lai, J. N. Culliney and S. Muroga developed the whole sets of algorithms for various kinds of transduction procedures [2][14][15][22] and then, they implemented the algorithms into the transduction programs [1][19][20][21]. K. C. Hu, K. R. Hohulin and B. Plagsiri then considered fan-in, fan-out and level restrictions into the transduction procedures [9][11][12][38]. L. G. Legge designed a transformation program which is able to transform a network into fan-in/fan-out restricted form [24].

The NETwork TRAnsduction (NETTRA) system was designed recently to organize the initial network programs (will be described in detail later), the transformation program and the transduction programs together so that anyone can use this system to design near-optimal NOR(NAND) networks under fan-in, fan-out and/or level restrictions.

The purpose of this report is to review the basic principles of the transduction procedures and explain how the NETTRA system is organized.

*The transduction methods are designed for finding near-optimal NOR networks. We can design a near-optimal NAND network for a given function by (1) designing a NOR network for the dual of the given function and (2) replacing each NOR gate with a NAND gate in the resulting NOR network. The transduction methods, hence, can also be used for near-optimal NAND network design. In this report, we will concentrate in discussion of NOR networks only.

The detailed contents of this report are as follows: Chapter 2 provides the outline of the system. It explains how the system treats a given problem (i.e., how to derive a near-optimal network for a given switching function (or a set of switching functions) under certain constraints specified by a user) and also describes what initial network methods and transduction procedures are included in the system. Chapter 3 gives primarily the review of basic principles of the transduction procedures. Section 3.1 introduces briefly the methods for finding initial networks (this part was never explained in detail in other reports). Sections 3.2 and 3.3 give summaries of basic principles for the fan-in/fan-out restricted transformation procedure and the transduction procedures. Chapter 4 provides the explanations of the functions of important subroutines, the detailed organization of the control subroutine MAIN, and the overlay structure of the system. Chapter 5 gives the statistics and the experimental results. The effects of different initial network methods and the effectiveness of different transduction procedures are compared. Four control sequences* are designed for the user's convenience. Chapter 6 provides the conclusions.

*The meaning of the control sequences will become clear later.

2. OUTLINE OF THE NETTRA-SYSTEM

The NETTRA-system can treat the following four types of problems:

(1) Find near-optimal networks for the given function(s) under no fan-in/fan-out restrictions or level restriction.

(2) Find near-optimal networks for the given function(s) under only fan-in/fan-out restrictions.

(3) Find near-optimal networks for the given function(s) under level restriction only.

(4) Find near-optimal networks for the given function(s) under both fan-in/fan-out restrictions and level restriction.

Since type (1) and type (3) problems can be considered as special cases of type (2) and type (4) problems, respectively, the approaches which can treat type (2) and type (4) problems can also solve type (1) and type (3) problems.

In this chapter, we introduce the outline of the NETTRA system primarily based on the approach for solving type (2) and type (4) problems.

The NETTRA system designs near-optimal networks for any given switching function(s) under only fan-in/fan-out restrictions (i.e., type (2) problem) according to the following procedures:

<u>Step 1</u>: Find an initial network* for the given function(s) by a conventional design method to be specified by a user as an

[*] As an alternative, the initial network designed by any other method can even be read into the NETTRA system.

option, ignoring fan-in/fan-out restrictions. Usually the initial network can be obtained very easily (i.e., in a very short time), but it may have many redundant gates and connections and it may not satisfy the given fan-in/fan-out restrictions.

Step 2: Apply the transduction procedures to simplify the initial network found in step 1 or the network obtained in step 4 without considering fan-in/fan-out restrictions. Usually the initial network is simplified, but the resultant network may not be fan-in/fan-out restricted. If the resultant network is fan-in/fan-out restricted, then we obtain a solution. Otherwise go to step 3.

Step 3: Employ the transformation procedure (developed by J. G. Legge) to transform the network obtained in step 2 into fan-in/fan-out restricted form.

Step 4: Apply the transduction procedures considering fan-in/fan-out restrictions, to simplify the fan-in/fan-out restricted network in step 3.

The resultant network after applying steps 1, 2, 3 and 4 is a near-optimal solution for the given problem. But the sequence of steps 2-3-4 can be applied repeatedly to try to simplify the network further.

If the given problem has both the fan-in/fan-out restrictions and level restriction (i.e., type (4) problem), the above 4 steps can still be used. But it is not guaranteed that a feasible network will be obtained by this approach even if there do exist optimal solutions for the given problem. Another approach for a problem with both fan-in/fan-out restrictions and level restriction imposed is explained below:

Step 1': Find a level-restricted initial network for the given problem. This initial network may not satisfy the fan-in/fan-out restrictions.

Step 2': Apply the fan-in/fan-out restricted and level-restricted transduction procedures to simplify the network obtained in step 1' without violating the level restriction.

If the resultant network after applying step 1' and step 2' satisfies fan-in/fan-out restrictions, then a feasible network has been obtained. A similar but more complex approach is implemented in [12] and is also included in the NETTRA system as an option (this will be explained in Chapter 4 and Chapter 5).

Several conventional methods are used in deriving initial networks for the NETTRA system. They are:

(1) Universal NOR network method [28][42].

(2) Three-level network method.

(3) Branch-and-bound method [37].

(4) Tison's method [3][31][40].

(5) Gimpel's algorithm [7].

(6)* Level-restricted initial network method [12].

Method (1) through method (5) are for fan-in/fan-out restricted problems. Each method has advantages and disadvantages and will be discussed in

*This method starts from the result obtained by Tison's method (4).

detail in Chapter 3. Method (6) is used only for level-restricted and fan-in/fan-out restricted problems.

There is one transformation procedure included in the system which can transform a non-fan-in/non-fan-out restricted network into fan-in/fan-out restricted form.

The transduction procedures are classified into the following five groups* according to their characteristics and capabilities:

(1) Pruning procedures [2][20].

(2) Procedures based on gate merging [19][22][38].

(3) Procedures based on gate substitution [2][19][20][22][38].

(4) Procedures based on connectable and disconnectable functions [1][9][14].

(5) Procedures based on error-compensation [11][15][21].

The basic principles and algorithms of these transduction procedures will be reviewed in Chapter 3.

The FORTRAN subroutines which realize the initial network methods, the transformation procedure and the transduction procedures are organized by the control subroutine MAIN. Figure 2.1 shows a general flowchart for the NETTRA system, where I/O supporting subroutines are used for inputing data, printing the results and punching the intermediate results.

*In the previous reports on the transduction procedures, (2), (3) and (4) are grouped as "General procedures."

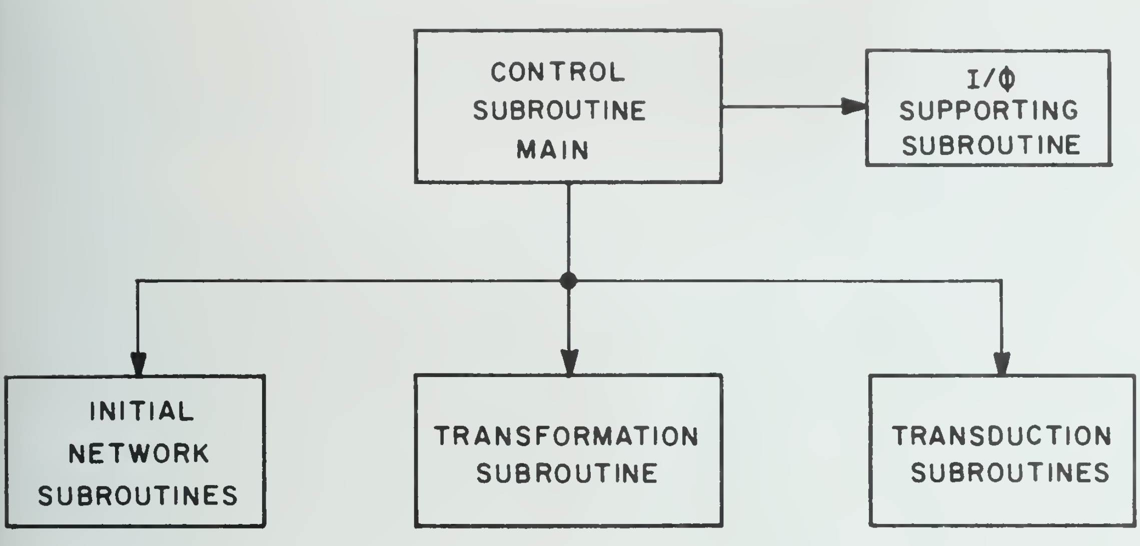

CONTROL SUBROUTINE MAIN
I/Φ SUPPORTING SUBROUTINE
INITIAL NETWORK SUBROUTINES
TRANSFORMATION SUBROUTINE
TRANSDUCTION SUBROUTINES

Fig. 2-1 General flowchart for the NETTRA-system.

3. BASIC IDEAS AND PRINCIPLES

3.1 Initial Network Synthesis

In the NETTRA system, there are currently six methods which can derive initial networks. Some methods produce initial networks in a very short time, and some methods take a longer time but produce initial networks with fewer gates and/or fewer connections. For any given switching function, if we start from different initial networks and apply the same transduction procedures, then we will usually get different results.

Since the initial network design methods have to be combined with the transduction procedures to obtain the near-optimal networks, it is not easy to judge which initial network design method is superior to others. The statistics in Chapter 5 gives some picture of the influence of initial networks on the final results.

3.1.1 Universal NOR Network Method [28]

Given an n-variable switching function f (with external input variables $x_1, x_2, \ldots, x_n$),* let $S = \{m_i | m_i$ is a minterm, $i = 0,1,2,\ldots,2^n-1\}$ (i.e., S is the set of all minterms) and let $M = \{m_i | m_i \in S \text{ and } f(m_i) = 1\}$. Then the function f can be expressed with the minterm expansion $\sum_{m_i \in M} m_i$.

As can easily be seen, f can also be expressed as $\overline{\sum_{m_i \in S-M} m_i}$. Hence if

* For the sake of convenience, assume that f is single-output and completely specified.

there exists an NOR network in which each gate realizes a function which is a minterm contained in S-M, then we can feed the outputs of these gates as inputs to another NOR gate to realize the function f at the output of this new NOR gate. The gates which are useless to realize this function can be eliminated (the elimination is actually done by the transduction methods).

The universal NOR network method can construct a NOR network consisting of 2^n gates with only uncomplemented external variables as network inputs. Each gate in this network realizes a function which is an element in S (i.e., a minterm). Therefore, any switching function f can be realized by the above approach with at most 2^n gates.* Figure 3.1.1-1 shows a 3-variable universal NOR network. The function realized by each gate is also shown.

The algorithm for generating the universal NOR network is presented below.

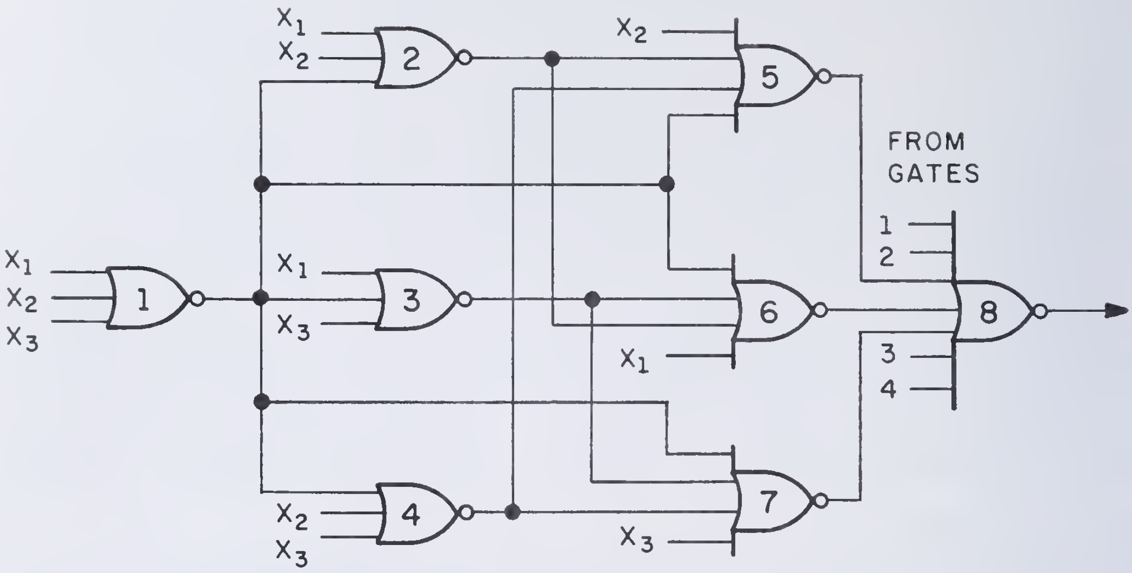
Let n be the number of external variables, and let the level number ℓ of a gate I be the maximum among the level numbers of gates along all paths connecting gate I and the output gate, which has gate level 1. In the n-variable universal NOR network, $1 \leq \ell \leq n+1$ must hold.

Algorithm for Generating the Universal NOR Network

(1) Connect all external variables to the highest level (the (n+1)-th level) gate.

(2) For each ℓ such that $1 < \ell < n+1$, find all possible sets of combinations of ℓ-1 external variables. Connect each set of ℓ-1 external variables to an ℓ-th level gate.

*The network requires exactly 2^n gates only when f is a one-minterm function.



$f_{g_1} = \bar{x}_1 \bar{x}_2 \bar{x}_3$ $\qquad$ $f_{g_5} = x_1 \bar{x}_2 x_3$

$f_{g_2} = \bar{x}_1 \bar{x}_3 x_3$ $\qquad$ $f_{g_6} = \bar{x}_1 x_2 x_3$

$f_{g_3} = \bar{x}_1 x_2 \bar{x}_3$ $\qquad$ $f_{g_7} = x_1 x_2 \bar{x}_3$

$f_{g_4} = x_1 \bar{x}_2 \bar{x}_3$ $\qquad$ $f_{g_8} = x_1 x_2 x_3$

Fig. 3.1.1-1 3 variable universal NOR network (f_{g_i} is the function realized at the output of gate i . The outputs of some of these gates are connected to an extra gate whose output is to realize a given function.)

(3) For each ℓ-th $(1 < \ell < n+1)$ level gate I, connect the outputs of all higher level gates which have external variable sets including the external variables to gate I, to gate I.

(4) For the output gate, connect the outputs of all higher level gates as its inputs.

The validity of the algorithm can easily be proved by the map factoring method for NOR network design [28][31], hence it is omitted here. A three-variable Karnaugh map and the corresponding loops derived by the map factoring method are shown in Fig. 3.1.1-2. Each shaded circle in Fig. 3.1.1-2 corresponds to a minterm which is shown in Fig. 3.1.1-1.

The following properties are observed with respect to the above algorithm:

(a) The number I_ℓ^n of gates in any level ℓ $(1 \leq \ell \leq n+1)$ is

$$I_\ell^n = \binom{n}{\ell-1} = \frac{n!}{(\ell-1)!\cdot(n-\ell+1)!}$$

(b) The total number of gates in the universal network is

$$\sum_{\ell=1}^{n+1} I_\ell^n = \sum_{\ell=1}^{n+1} \binom{n}{\ell-1} = \sum_{\ell=1}^{n+1} \frac{n!}{(\ell-1)!(n-\ell+1)!} = 2^n$$

The following example describes how to realize a given function by utilizing the universal network.

<u>Example 3.1.1-1</u> - Consider the function $f(x_1,x_2,x_3) = \Sigma(0,2,3,4,6,7)$. Since the set S-M = $\{1,5\}$, we can feed the outputs of gates 2 and 5 to another gate (gate 9) in Figure 3.1.1-1 to realize f. The result is shown in Fig. 3.1.1-3.

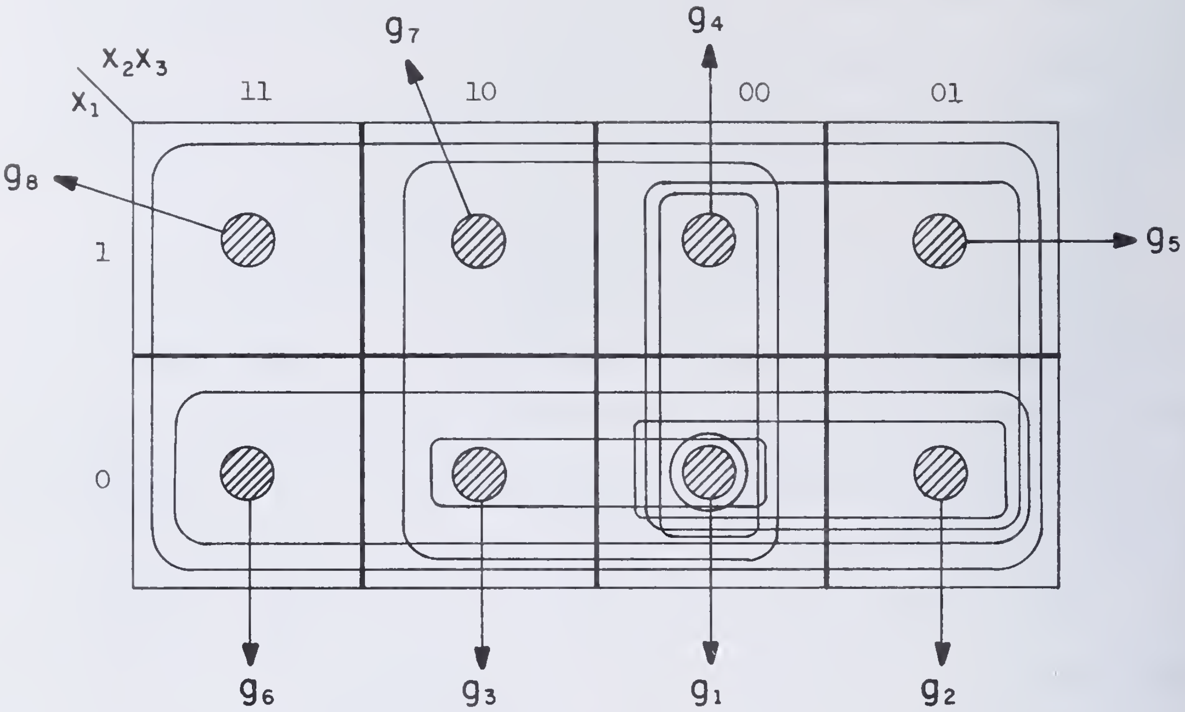


Fig. 3.1.1-2 The 3-variable Karnaugh map with loops and shaded circles for proving the algorithm for generating the universal network by using map factoring method.

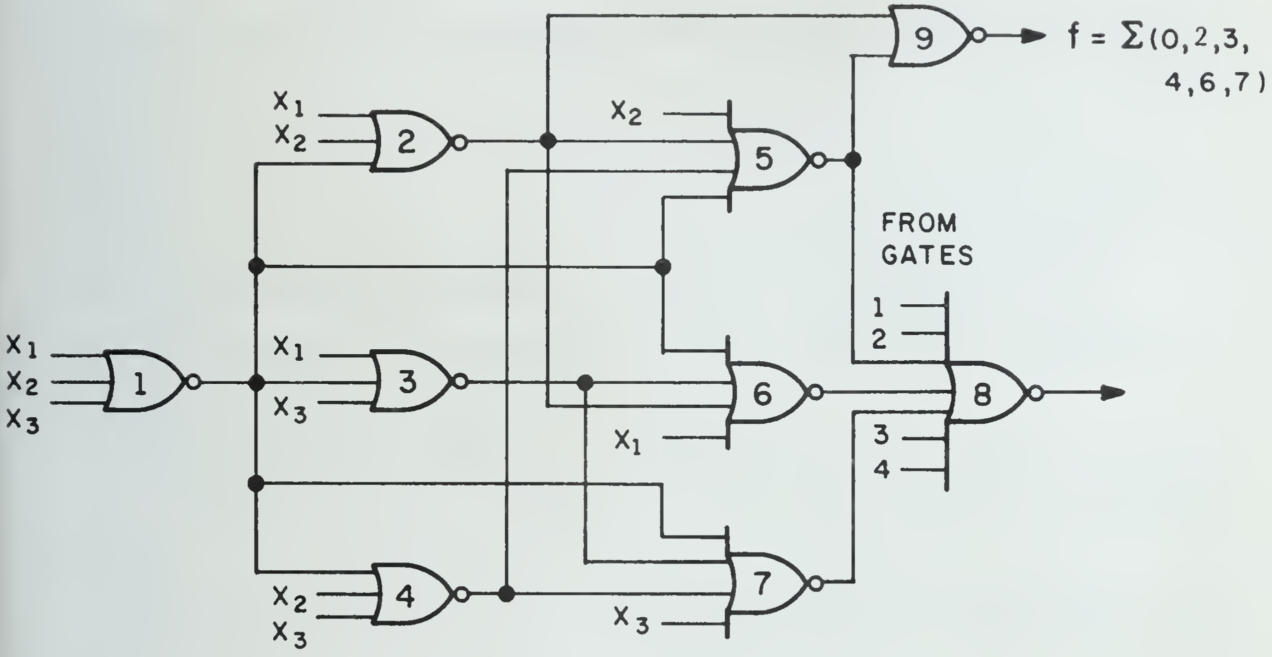


Fig. 3.1.1-3   Example 3.1.1-1

It is easy to find that there are many redundant gates and connections in Fig. 3.1.1-3. For example, gates 3, 6, 7 and 8 are useless for realizing f and hence can be removed. The transduction procedures can then be applied to remove redundant gates and connections as many as possible.

3.1.2 Three-Level Network Method

It is mentioned in section 3.1.1 that any switching function f can be expressed as $\overline{\sum_{m_i \in S-M} m_i}$, where S is the set of all minterms and M is the set of minterms which corresponds to 1-vectors. (An input vector $\vec{x}$ is called a 1-vector if $f(\vec{x}) = 1$; it is called a 0-vector if $f(\vec{x}) = 0$.) The three-level network method developed by Y. Kambayashi is also based on the above idea to design a NOR network with only uncomplemented external variables as inputs, but the networks designed by this method are restricted to those with at most three levels. Before presenting the algorithm, let us introduce the following definitions:

Let V_n be the set of all n-dimensional binary (0 or 1) vectors, where n is the number of external variables.

Definition 3.1.2-1 - Given two input vectors $\vec{X}_1 = (x_{11},\ldots,x_{1n})$ and $\vec{X}_2 = (x_{21},\ldots,x_{2n})$, then $\underline{\vec{X}_1 < \vec{X}_2}$ if $x_{1i} \le x_{2i}$ for every $i = 1,\ldots,n$ but $x_{1i} < x_{2i}$ for at least one i. $\vec{X}_1$ and $\vec{X}_2$ are said to be equal $(\vec{X}_1 = \vec{X}_2)$ if $x_{1i} = x_{2i}$ for every $i = 1,\ldots,n$. $\vec{X}_1$ and $\vec{X}_2$ are said to be incomparable if none of $\vec{X}_1 < \vec{X}_2$, $\vec{X}_2 < \vec{X}_1$ and $\vec{X}_1 = \vec{X}_2$ holds.

Definition 3.1.2-2 - The weight $W(\vec{X})$ of an input vector $\vec{X}$ is the number of ones in $\vec{X}$. The distance between two input vectors $\vec{X}_1$ and $\vec{X}_2$ is $d(\vec{X}_1,\vec{X}_2) = W(\vec{X}_1 \oplus \vec{X}_2)$, where $\vec{X}_1 \oplus \vec{X}_2 = (x_{11} \oplus x_{21}, \ldots, x_{1n} \oplus x_{2n})$.

Definition 3.1.2-3 - A Z-set is a maximal set of input vectors (i.e., no other Z-set is a proper subject of this Z-set) such that the following conditions are satisfied:

(1) In each Z-set, all vectors must be 0-vectors and there exists one and only one 0-vector which is greater than any other 0-vectors.

(2) 0-vectors $\vec{X}_1$ and $\vec{X}_2$ do not belong to the same Z-set if $\vec{X}_1 < \vec{X}_2$ and there exists another 1-vector $\vec{X}_3$ such that $\vec{X}_1 < \vec{X}_3 < \vec{X}_2$ holds. $\vec{X}_1$ $\vec{X}_2$ belong to the same Z-set if the above condition does not hold.

(3) In each Z-set, there exists at least one vector $\vec{X}$ which does not belong to any other Z-set.

A Z-set, in which $\vec{X}^*$ is greater than any other vectors, is denoted by $Z(\vec{X}^*)$. $\vec{X}^*$ is called the root of $Z(\vec{X}^*)$.

Definition 3.1.2-4 - A Y-set is a maximal set of input vectors such that the following conditions are satisfied:

(1) In each Y-set, all vectors must be 1-vectors and there exists one and only one 1-vector which is greater than any other 1-vectors.

(2) 1-vectors $\vec{X}_1$ and $\vec{X}_2$ do not belong to the same Y-set if $\vec{X}_1 < \vec{X}_2$ and there exists another 0-vector $\vec{X}_3$ such that $\vec{X}_1 < \vec{X}_3 < \vec{X}_2$ holds. $\vec{X}_1$ and $\vec{X}_2$ belong to the same Y-set if the above condition does not hold.

(3) In each Y-set, there exists at least one vector $\vec{X}$ which does not belong to any other Y-set.

A Y-set, in which $\vec{X}^*$ is greater than any other vectors, is denoted by $Y(\vec{X}^*)$. $\vec{X}^*$ is called the root of $Y(\vec{X}^*)$.

Example 3.1.2-1 - In Fig. 3.1.2-1, the given four-variable switching function is $f = \overline{\Sigma(2,5,6,7,9,13,14,15)}$. There are 3 Z-sets and 3 Y-sets according to the previous definitions.

$Z(1111) = \{(1111),(1110),(0110),(1101),(0101),(0111)\}$

$Z(1101) = \{(1101),(1001),(0101)\}$

$Z(0110) = \{(0110),(0010)\}$

$Y(1011) = \{(1011),(1010),(0011)\}$

$Y(1100) = \{(1100),(1000),(0100),(0000)\}$

$Y(0011) = \{(0011),(0001)\}$

Notice that $\{(0000),(0001)\}$ is not a Y-set because $(0000) \in Y(0011)$ violates condition (iii) of Definition 3.1.2-4. Also any subset of the Y-sets (or the Z-sets) shown above is not a Y-set (or a Z-set).

In the three-level initial network method (the output gate is the first level gate), each Y-set may correspond to a third-level gate. In order to know which third-level gates should feed which second-level gates, the "adjacent relations" among the Y-sets and the Z-sets is introduced below.

Definition 3.1.2-5 - $Y(\vec{X}_1)$ is said to be adjacent to $Z(\vec{X}_2)$ if (1) there exist a $\vec{X}_3 \in Y(\vec{X}_1)$ and a $\vec{X}_4 \in Z(\vec{X}_2)$ such that $d(X_3,X_4) = 1$ and the $\vec{X}_3 < \vec{X}_4$ and (2) for every pair $\vec{X}_3 \in Y(\vec{X}_1)$ and $\vec{X}_4 \in Z(\vec{X}_2)$, no $\vec{X}_4 < \vec{X}_3$ holds.

Example 3.1.2-2 - In Fig. 3.1.2-1, consider the Z-set Z(1111) and Y-set Y(1100) first. Since the 1-vector $(1100) \in Y(1100)$, the 0-vector $(1110) \in Z(1111)$, $(1100) < (1110)$ and $d(1100,1110) = 1$, condition (1) of the previous definition is satisfied. Besides, no vector which belongs to Z(1111) is less than (<) any vector which belongs to Y(1100), so condition (2) is also satisfied. Hence the Y-set Y(1100) is adjacent to the Z-set Z(1111). Similarly, Y(1011) and Y(0011) are found to be adjacent to Z(1111).

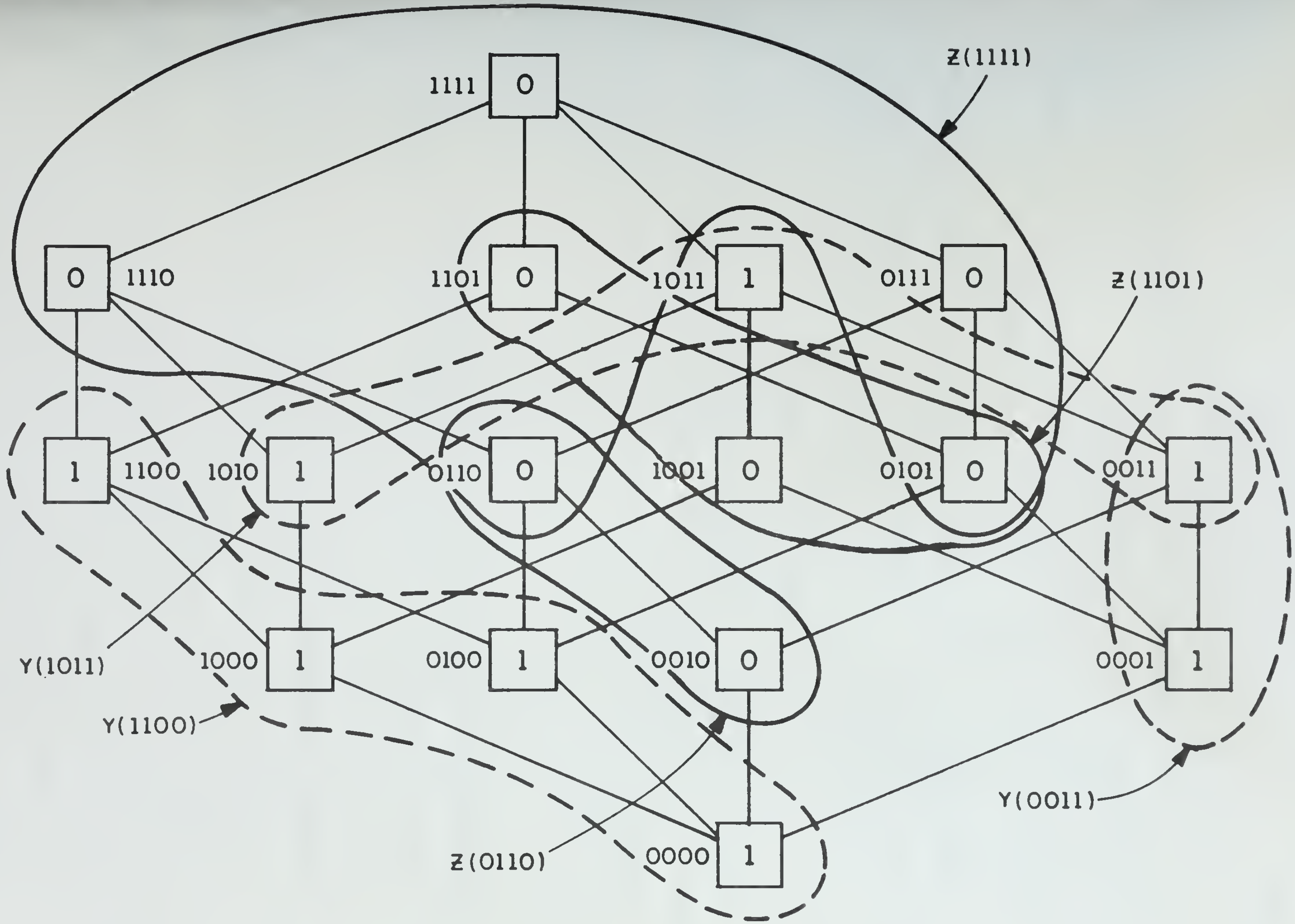


Fig. 3.1.2-1 Example 3.1.2-1

The adjacency relations for the given example can be summarized as follows:

Y(0011) is adjacent to Z(1111) and Z(1101).

Y(1100) is adjacent to Z(1111), Z(1101) and Z(0110).

Y(1011) is adjacent to Z(1111) only.

Notice that each Y-set (or Z-set) does not have to be adjacent to some Z-sets (or Y-sets). The following is such an example.

Example 3.1.2-3 - In Fig. 3.1.2-2, the given function is the 3-variable parity function. There are 4 Y-sets and 4 Z-sets and each Y-set or Z-set consists of only one vector. Apparently, Z(000) is not adjacent to any Y-set and Y(111) is not adjacent to any Z-set.

Definition 3.1.2-6 - The β-set of an input vector $\vec{X}$ is the set of all uncomplemented external variables which appears as zeros in $\vec{X}$.

Example 3.1.2-4 - The β-set of 1100 is $\{x_3, x_4\}$; the β-set of 1111 is the empty set ∅.

Now we are ready to present the algorithm.

Algorithm for generating the three-level NOR network:

Step 1: Find all Z-sets and Y-sets for the given function f and then calculate the adjacency relations.

Step 2: Construct a third-level gate for each $Y(\vec{X})$ which is adjacent to some Z-sets. The inputs for this gate are the elements (external variables) of the β-set of the $\vec{X}$.

Step 3: Construct a second-level gate for each $Z(\vec{X})$. The inputs for this gate are the elements of the β-set of the $\vec{X}$ and the outputs of those third-level gates which correspond to the Y-sets which are adjacent to $Z(\vec{X})$. If $Y(\vec{X}_1)$ and $Y(\vec{X}_2)$ are adjacent to $Z(\vec{X})$ and

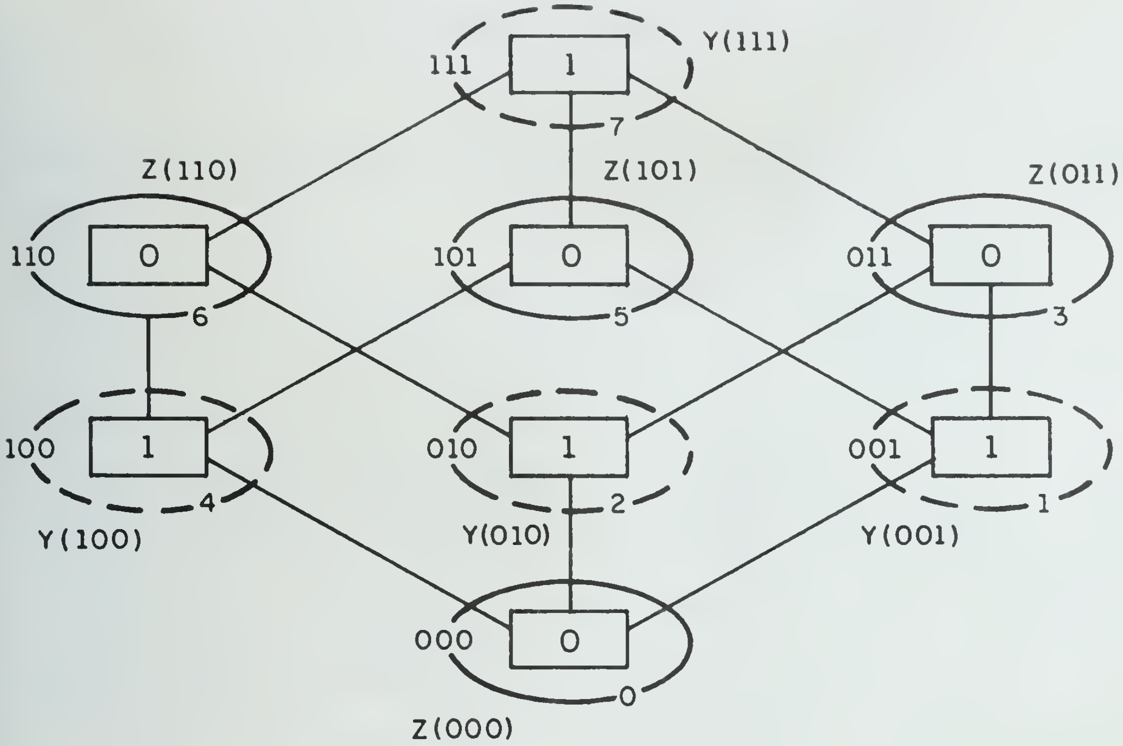


Fig. 3.1.2-2   Example 3.1.2-3

$\vec{X}_1 > \vec{X}_2$, then the corresponding third-level gate for $Y(\vec{X}_2)$ is not necessary to be connected to the corresponding second-level gate for $Z(\vec{X})$.

Step 4: Feed the outputs of all second-level gates to the first-level gate (the output gate). Stop.

Example 3.1.2-5 - Following the above algorithm, a three-level network for Example 3.1.2-1 is obtained in Fig. 3.1.2-3.

In Fig. 3.1.2-3, the outputs of gates 1, 2, and 3 correspond to the Y-sets Y(1100), Y(0011), and Y(1011), respectively; and the outputs of gates 4, 5 and 6 correspond to the Z-sets Z(0110), Z(1101) and Z(111), respectively. Since Y(1011) and Y(0011) are adjacent to Z(1111) and (0011) < (1011), the output of gate 2 (corresponding to Y(0011) is not connected to gate 6 (corresponding to Z(1111)) according to step 3.

In the above algorithm, the function realized at the outputs of each second-level gate is the disjunction of minterms which correspond to 0-vectors in the corresponding $Z(\vec{X})$. The input of the first-level gate is, therefore, the disjunction of all minterms which correspond to all 0-vectors of f. The detailed proof is omitted here.

An interesting phenomenon which arises in the previous algorithm is that we cannot get a network whose output gate has external variables as inputs. The following is an example.

Example 3.1.2-5 - Give the 3-variable function $f = \Sigma(0,3)$ in Fig. 3.1.2-4(a) and (b), the three-level network is obtained by applying the algorithm. Apparently, gate 2 and gate 5 are redundant and can be removed; i.e., we can connect x_1 to gate 6 directly.

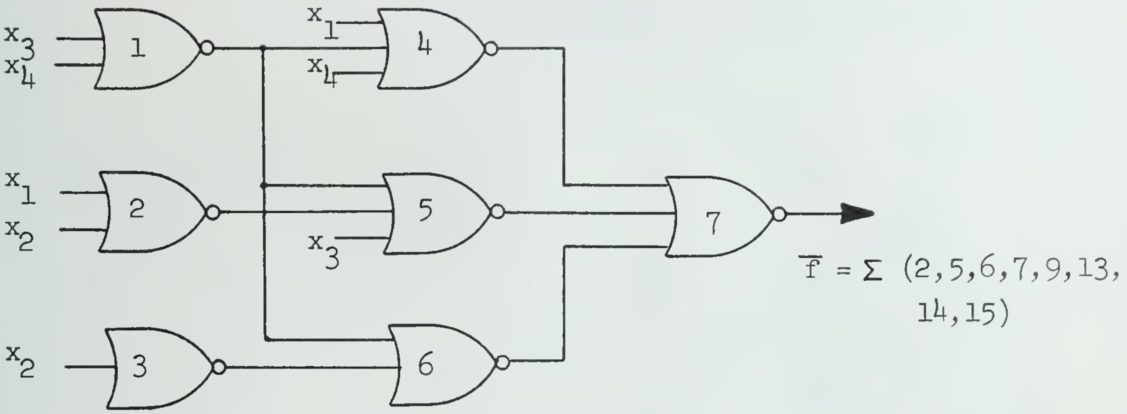
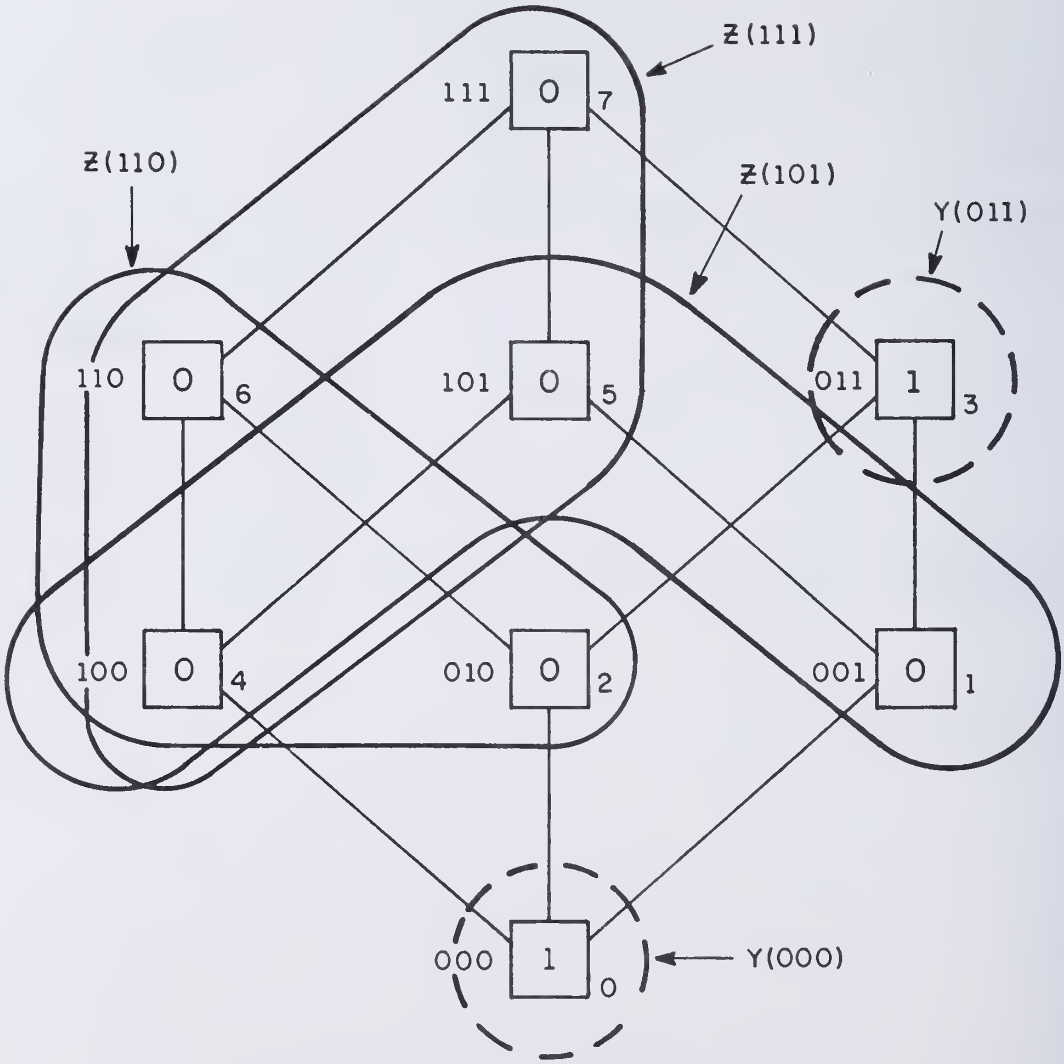
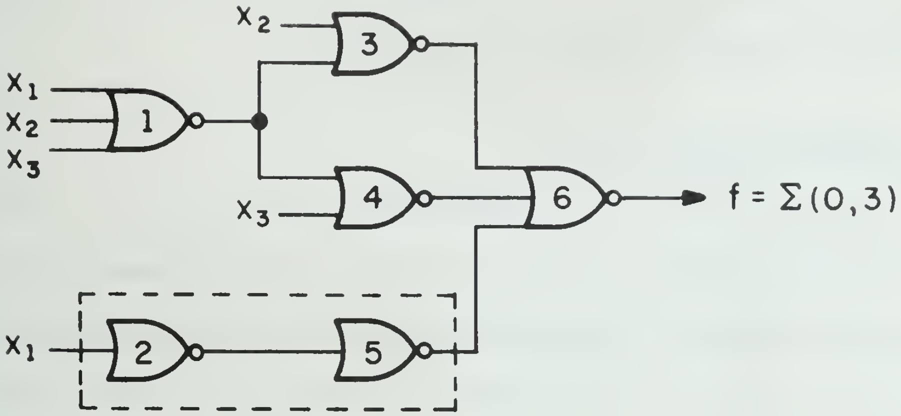


Fig. 3.1.2-3 A three-level network for Example 3.1.2-1.


Ƶ(111)
Ƶ(110)
Ƶ(101)
Y(011)
Y(000)
111
0
7
110
0
6
101
0
5
011
1
3
100
0
4
010
0
2
001
0
1
000
1
0

(a)



(b)

Fig. 3.1.2-4 Example 3.1.2-5. Gate 2 and gate 5 are redundant

The above algorithm for generating the three-level network does not guarantee the minimality of the numbers of gates and connections. But because of the simplicity of the algorithm, it is good for finding initial networks for the NETTRA system.

3.1.3 Branch-and-bound Method

The branch-and-bound algorithm was applied by E. S. Davidson [4] to the synthesis problem of optimal combinational networks for arbitrary switching functions using NAND gates, by introducing the desirability order and other speed improvement gimmicks. T. Nakagawa and H. C. Lai implemented Davidson's algorithm for NOR gates by using simpler heuristics than Davidson's and also by incorporating a new gimmick called "redundancy check" [37]. The branch-and-bound algorithm used in designing initial networks in the NETTRA system is the simplified version of Nakagawa and Lai's algorithm, it only finds the first feasible solution but doesn't try to obtain the optimal solutions for the given function.

Let x_ℓ, $\ell = 1,\ldots,n$, be n external variables, and let f be the given function.* The x_ℓ, $\ell = 1,\ldots,n$, and f are expressed in the following way:

$$x_\ell = (x_\ell^0,\ldots,x_\ell^{2^n-1}), \quad \ell = 1,\ldots,n$$

$$f = (f^0,\ldots,f^{2^n-1})$$

*Again, for the sake of convenience, only the single-output network case is considered here.

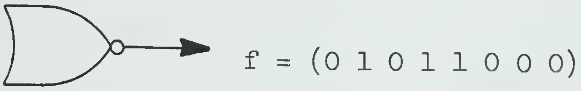
For example, if $f = \bar{x}_1 x_3 \vee x_1 \bar{x}_2 \bar{x}_3$, then

$$x_1 = (00001111)$$
$$x_2 = (00110011) \qquad (3.1.3\text{-}1)$$
$$x_3 = (01010101)$$

and

$$f = (01011000)$$

Let us assign an NOR gate to f, as shown in Fig. 3.1.3-1. The output of this NOR gate is expected to eventually realize the given f, though at this stage the gate does not realize f yet since no inputs are connected. This network with a single gate is called the <u>initial solution</u>.



<u>Fig. 3.1.3-1</u> The initial solution to $f = \bar{x}_1 x_3 \vee x_1 \bar{x}_2 \bar{x}_3$. The output of the NOR gate is assigned f, but the gate has no inputs yet.

The algorithm starts with the initial solution, and expands the initial solution by connecting external variables, by introducing new gates, or by making interconnections among gates, so that the resulting loop-free network realizes the given function.

In accordance with (3.1.3-1), the output of each gate to be introduced into the network is represented in the form of 2^n-tuple as $(P^0, \ldots, P^{2^n-1})$, where each P^j for $j = 0, \ldots, 2^n-1$, may assume the values 0 or 1. It should be noted, however, that the algorithm will not assign a definite value 0 or 1 at once to all components, P^j's, of a gate. Accordingly, we use the symbol * to denote that the value of P^j is unassigned. Let us find out a necessary condition that the output $(P^0, \ldots, P^{2^n-1})$ of any gate in an NOR network satisfies. Consider two gates, i and k. Let $(P_i^0, \ldots, P_i^{2^n-1})$ and $(P_k^0, \ldots, P_k^{2^n-1})$ denote the outputs of gate i and gate k, respectively. If gate i is connected to gate k, then the components P_i^j and P_k^j must satisfy the following condition, no matter whether or not gate i and gate k have other inputs, because the gates perform NOR operation:

$$\left.\begin{array}{lll} & P_k^j = 0 & \text{for all } j \text{ such that } P_i^j = 1 \\ \\ \text{and} & P_i^j = 0 & \text{for all } j \text{ such that } P_k^j = 1 \end{array}\right\} \qquad (3.1.3\text{-}2)$$

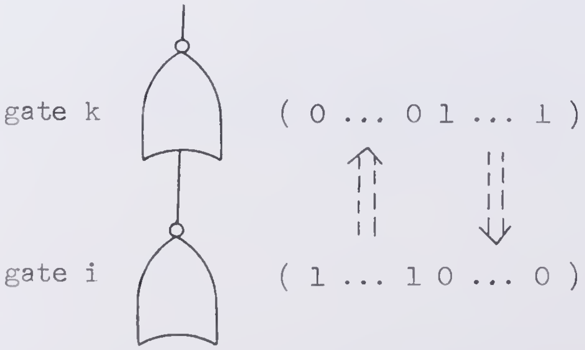


Fig. 3.1.3-2 If there are 1-components in the output of an NOR gate, then corresponding components in its immediately preceding/succeeding gates must be 0.

A similar condition must also hold between the output of gate k and an external variable x_ℓ, when x_ℓ is connected to gate k, regardless of other inputs to gate k:

$$\left.\begin{array}{lll} & P_k^j = 0 & \text{for all } j \text{ such that } x_\ell^j = 1, \\ \text{and} & x_\ell^j = 0 & \text{for all } j \text{ such that } P_k^j = 1 \end{array}\right\} \qquad (3.1.3\text{-}3)$$

If the assignment of binary values to the components P^j's of the output $(P^0,\ldots,P^{2^n-1})$ of a gate satisfies the above condition with respect to all of immediately preceding/succeeding gates and all connected external variables, then we call this assignment of $(P^0,\ldots,P^{2^n-1})$ a <u>feasible assignment</u>. (Notice that some of components P^j's could be *.)

Using the concept of feasible assignment, let us define an "intermediate solution."

<u>Definition 3.1.3-1</u> - A network of R gates, $R \geq 1$, with external variables x_ℓ is called an <u>intermediate solution</u> if the network satisfies the following set of conditions:

(i) The entire network has no loops. R gates are numbered 1 through R, as gate 1, ..., gate R.

(ii)-a The first gate (i.e., gate 1) is assigned the output function.

(ii)-b The outputs of the remaining gates (i.e., gate i, $i = 2,\ldots,R$), if any, are completely or incompletely specified. Each gate i for $i = 2,\ldots,R$, is connected to at least one of other gates in the network.

(iii) The assignment of the output of each gate is feasible.

Notice that the initial solution defined previously is a special case of an intermediate solution.

An intermediate solution network may or may not realize the given function. An intermediate solution whose network realizes the given function is said to be a feasible solution. (A feasible solution whose cost is the least among all feasible solutions is an optimal solution, though we do not intend to obtain it.)

Definition 3.1.2-2 - A component $P_k^{j_o} = 0$ of gate k is said to be covered, if gate k has at least one input (i.e., the output of gate i or external variable x_ℓ) whose j_o-th component ($P_i^{j_o}$ or $x_\ell^{j_o}$) is 1. $P_k^{j_o} = 0$ of gate k is said to be uncovered, if $P_k^{j_o}$ is not yet covered.

Fig. 3.1.3-3 is an example of intermediate solution, where some components are already covered. (The covered components are shown with underlines.)

Clearly an intermediate solution is a feasible solution if all output components $P_k^j = 0$ in all gates k are covered.

Definition 3.1.3-3 - Suppose $P_k^{j_o}$ in gate k is an uncovered component in a given intermediate solution. $P_k^{j_o}$ can be covered either by an external variable or by a gate if the external variable or the gate is a possible cover, as follows:

In the NETTRA-system, the "cost" of a network is defined by $C = A \times R + B \times I$, where R is the number of gates, I is the total number of inputs to gates (i.e., the sum of connections of external variables and interconnections among gates), and A, B are non-negative coefficients (i.e., weights). Different combinations of weights A and B imply different design problems. For example, $A > 0$ and $B = 0$ implies finding a network with as few gates as possible, not counting the number of connections, $A \gg B > 0$ implies finding a network with as few connections as possible after the number of gates is reduced to as few as possible.

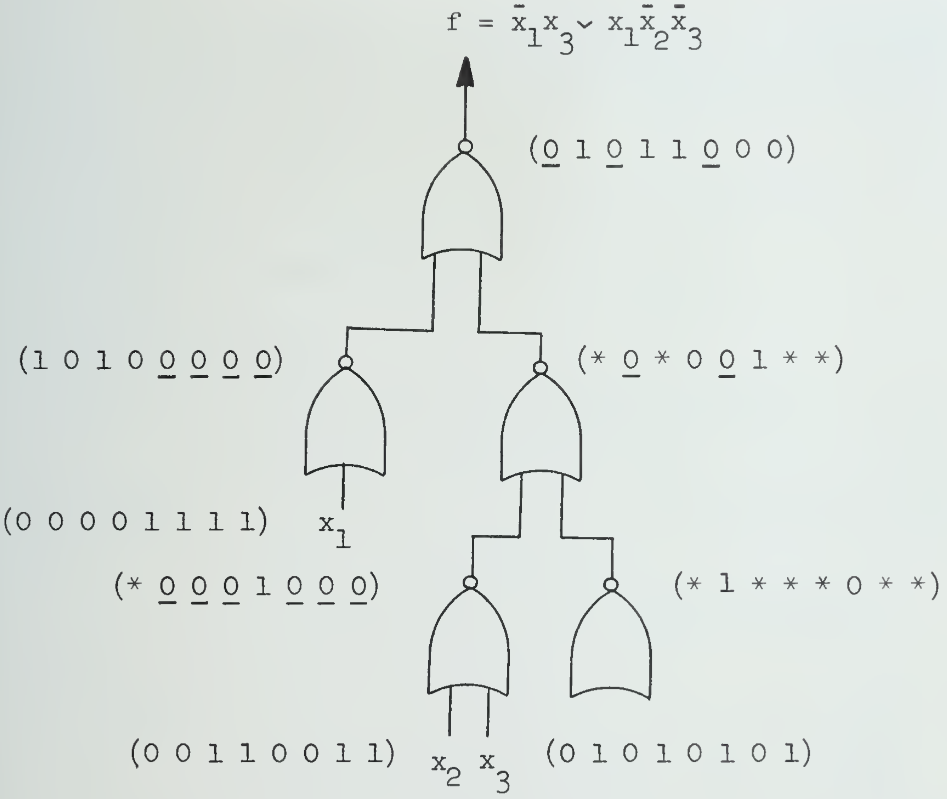

$f = \bar{x}_1 x_3 \vee x_1 \bar{x}_2 \bar{x}_3$
$(\underline{0}\ 1\ \underline{0}\ 1\ 1\ \underline{0}\ 0\ 0)$
$(1\ 0\ 1\ 0\ \underline{0}\ \underline{0}\ \underline{0}\ \underline{0})$
$(*\ \underline{0}\ *\ 0\ \underline{0}\ 1\ *\ *)$
$(0\ 0\ 0\ 0\ 1\ 1\ 1\ 1)$
x_1
$(*\ \underline{0}\ \underline{0}\ \underline{0}\ 1\ \underline{0}\ \underline{0}\ \underline{0})$
$(*\ 1\ *\ *\ *\ 0\ *\ *)$
$(0\ 0\ 1\ 1\ 0\ 0\ 1\ 1)$
x_2
x_3
$(0\ 1\ 0\ 1\ 0\ 1\ 0\ 1)$

Fig. 3.1.3-3 Example of an intermediate solution for $f = \bar{x}_1 x_3 \vee x_1 \bar{x}_2 \bar{x}_3$, where some components are covered and others are not(the covered components are underlined).

(I) An external variable x_ℓ which is not yet connected to gate k is a possible cover of $P_k^{j_o} = 0$ if x_ℓ satisfies the following condition:

$$\begin{cases} x_\ell^{j_o} = 1, & \text{and} \\ x_\ell^{j} = 0 & \text{for all } j \text{ such that } P_k^j = 1. \end{cases}$$

(II) A gate i which is already connected to gate k is a possible cover of $P_k^{j_o} = 0$ if gate i satisfies the following condition:

$$P_i^{j_o} = *.$$

(III) A gate i which is not yet connected to gate k is a possible cover of $P_k^{j_o} = 0$ if gate i satisfies the following condition:

$$\begin{cases} \text{a connection of gate i to gate k will not} \\ \text{form any loops,} \\ P_i^{j_o} = 1 \text{ or } *, \text{ and} \\ P_i^{j} = 0 \text{ or } * \quad \text{for all } j \text{ such that } P_k^j = 1. \end{cases}$$

(IV) A gate which is not yet incorporated in the intermediate solution is a possible cover. This gate is called <u>a new gate</u>, and satisfies the following condition:

The ouptut components are all *.

Sometimes there exists more than one possible cover for an uncovered component. In order to treat all possible covers and enumerate all intermediate solutions systematically, the following definitions are introduced.

Definition 3.1.3-4 - The selection criterion of uncovered components (SCUC) is the criterion under which an uncovered component $p_k^j = 0$ is selected from an intermediate solution under consideration.

Definition 3.1.3-5 - The implementation priority of-possible covers (IPPC) is the priority under which the order of implementation among the possible covers for the selected uncovered component is determined.

Definition 3.1.3-6 - The cost ceiling, or the incumbent cost, $\overline{C}$ is used to preclude all intermediate solution whose cost exceeds the cost of the current best feasible solution.*

The basic form of the algorithm is given below.

The branch-and-bound algorithm for finding the initial networks:

Step 0 (start): $k = 1$

Let S_1 denote the initial solution.

Set $\overline{C}$ to A × 50 + B × 99.†

Step 1: Calculate the cost C_k of the current intermediate solution S_k. Compare C_k with $\overline{C}$. If C_k is greater than $\overline{C}$, then go to step 7; otherwise go to step 2.

Step 2: Search for an uncovered component in S_k. If there is none, then go to step 8; otherwise go to step 3.

Step 3: Select one uncovered component from S_k, according to the selection criterion of uncovered component (SCUC). Let $\hat{P}$ denote it.

*The branch-and-bound algorithm for finding an initial network which is used in the NETTRA system is a simplified version of the algorithm for finding the optimal networks. Since only the first feasible solution is required, the cost ceiling $\underline{C}$ is set to the value such that all intermediate solution networks whose number of gates and connections exceed the possible maximal values (restricted by the core available) are precluded.

†This means that the numbers of gates and connections in any network is restricted to be at most 50 and 99, respectively.

Step 4: Make a list of all possible covers of $\hat{P}$.

Step 5: Select one possible cover from the list, according to the implementation priority of possible covers (IPPC).

Step 6: Increment k by 1.

Implement the possible cover selected at step 5, generating the augmented intermediate solution, S_k.

Go to step 1.

Step 7: The cost of the intermediate solution network is greater than $\overline{C}$. Stop.

Step 8: Print the Cost C_k and the feasible solution S_k. Stop.

The detailed discussion of the selection criterion of uncovered components (SCUC), the implementation priority of possible covers (IPPC), the speed improvement gimmicks, the redundancy checks, and the algorithm for finding the optimal networks are given in [37].

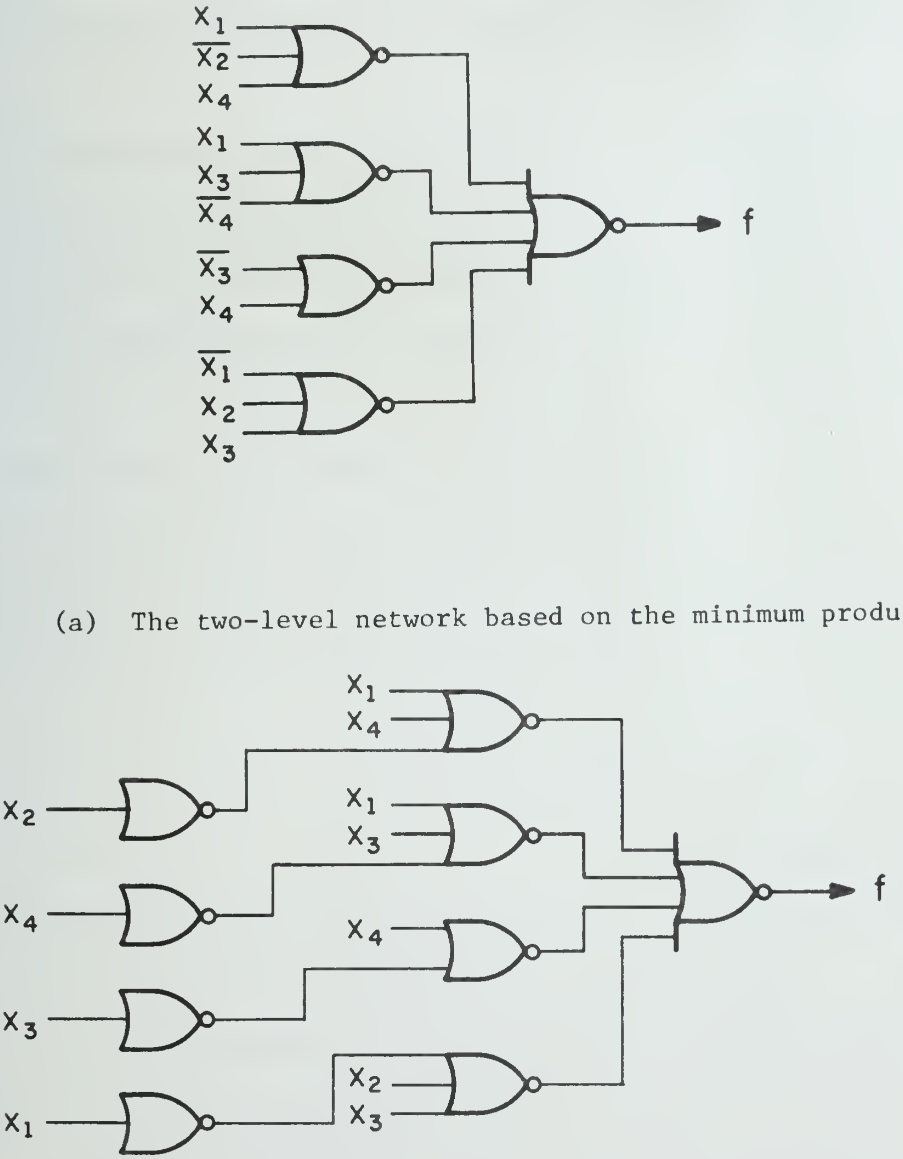
It should be noticed that the branch-and-bound method (outlined in this section) used in the NETTRA system can also design networks for incompletely specified and multiple-output functions with both complemented and uncomplemented variables or only uncomplemented variables as inputs.

3.1.4 Tison's Method

We can construct a two-level (if both complemented and uncomplemented external variables are permitted as inputs) or a three-level (if only complemented external variables are permitted as inputs) network based on each minimum product* for a given switching function. The following is an example to show this.

*The formal definitions for minimum product and minimum sums will be given later.

Example 3.1.4-1 - The four-variable function $f = \Sigma\ (0,3,7,11,12,13,15)$ has a minimum product $(x_1 \vee \bar{x}_2 \vee x_4)\ (x_1 \vee x_3 \vee \bar{x}_4)\ (\bar{x}_3 \vee x_4)\ (\bar{x}_1 \vee x_2 \vee x_3)$. The corresponding two-level and three-level networks are shown in Fig. 3.1.4-1(a) and Fig. 3.1.4-1(b), respectively.



(a) The two-level network based on the minimum product

(b) The three-level network based on the mininimum product

Fig. 3.1.4-1 Example 3.1.4-1

Obviously, no gate or connection can be removed from the networks based on minimum products without changing the outputs. Besides, according to experience, the networks based on the minimum products usually have low costs. Therefore if we can find some efficient way to generate the minimum products for the given function, then we can obtain economical intial networks easily.

Tison's method [40] is one way to obtain the minimum sums for a given function. Since the minimum products can be obtained by (1) finding the minimum sums for the dual of the given function, and then (2) by exchanging conjunction and disjunction in the minimum sums, we can apply Tison's method to design initial networks.

Now let us review Tison's method for finding the minimum sums. The following definitions are introducted first to facilitate later discussions.

Definition 3.1.4-1 - Let two switching functions be $f(\vec{x})$ and $g(\vec{x})$. If every $\vec{x}$ satisfying $f(\vec{x}) = 1$ satisfies also $g(\vec{x}) = 1$ but the converse does not necessary hold, we write

$$f(\vec{x}) \subseteq g(\vec{x}).$$

and we say that f implies g.

Definition 3.1.4-2 - An implicant of a switching function f is a term which implies f. An implicate of f is an alterm implied by f.

Definition 3.1.4-3 - A term P is said to subsume another term Q, if all the literals of Q are factors of P. Similarly an alterm P is said to subsume another alterm Q if all the literals of Q are among literals of P.

Definition 3.1.4-4 - A prime implicant of f is defined as an implicant of f such that no other term subsumed by it can be of an implicant of f. A prime implicate of f is defined as an implicate of f such that no other alterm subsumed by it can be an implicate of f.

Definition 3.1.4-5 - A variable whose complemented and uncomplemented literals both appear in a disjunctive form $f= P \vee Q \vee \ldots \vee T$ of a switching function f is called a biform variable. If only one of the literals for a variable appears, the variable is called a monoform variable.

Definition 3.1.4-6 - Assume two terms, P and Q given. If there is exactly one variable, say x, appearing without inversion in one term and with inversion in the other, in other words, if $P = xP'$ and $Q = \overline{x}Q'$ (no other variables appear with complement in one of P' and Q' without complement in the other), then the product of all literals except the literals of x, i.e., P'Q' is called the consensus of two terms, P and Q. Assume two alterms, V and W given. If there is exactly one variable, say x, appearing without inversion in one alterm and with inversion in the other, i.e., if $V = x \vee V'$ and $W = \overline{x} \vee W'$ where V' and W' are free of literals of x, then the disjunction of all literals except those of x, i.e., $V' \vee W'$ with duplicate literals deleted is called the consensus of two alterms, V and W.

Definition 3.1.4-7 - An irredundant disjunctive form for f is a disjunction of prime implicants such that removal of any of them makes the remaining expression not equivalent to the original f. An irredundant conjunctive form is a conjunction of prime implicates such that removal of any of them makes the remainder not equivalent to f.

An irredundant disjunctive form (also an irredundant conjuctive form) for a function is not necessarily unique.

Definition 3.1.4-8 - Prime implicants which appear in every irredundant disjunctive form for f are called essential prime implicants of f. Prime implicants which do not appear in any irredundant disjunctive form for f are called absolutely eliminable prime implicants of f. Prime implicants which appear in some irredundant disjunctive forms for f but not in all are called *conditionally eliminable prime implicants* of f. Essential prime implicates, absolutely eliminable prime implicates, and conditionally eliminable prime implicates are similarly defined.

Definition 3.1.4-9 - Among irredundant disjunctive forms of f, choose these with a minimum number of prime implicants. Among them, those with a minimum number of literals are called the *minimal sums*, or the *minimal disjunctive forms* for f. The *minimal products* (or *minimal conjuctive forms*) are irredundant conjunctive forms of f with a minimum number of literals, among irredundant conjunctive forms of f with a minimum number of prime implicates.

Definition 3.1.4-10 - The disjunction of all prime implicants of a switching function f is called *the complete sum* or *the all prime implicants disjunction*.

Tison's method for finding the minimum sums for a given function f consists of two major steps:

Step I: Find all prime implicants of f.

Step II: Find all irredundant disjunctive forms of f using the prime implicants found in step I.

The following two procedures (Procedure I and Procedure II) realize step I and step II, respectively.

Procedure I *(Tison method for the derivation of prime implicants)*: Assume that a function f is given in a disjunctive form $f = P \vee Q \vee \ldots \vee T$, where

P, Q, ..., T are products, and that we want to find all prime implicants of f. Denote the set of P, Q, ..., T, with S.

(1) Among the set of P, Q, ..., T, first delete every term subsuming another from the given expression. Find biform variables. Then choose one of them.

(2) For each pair of products, i.e., one with the complemented literal of the chosen variable and the other with the uncomplemented literal of that variable, generate the consensus.

Add the generated consensus to S. From S, delete every product which subsumes another.

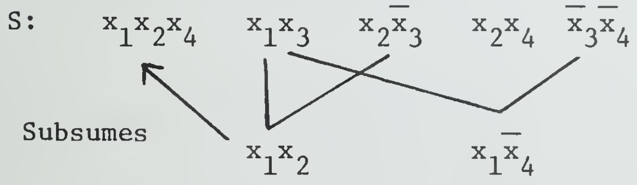
(3) Choose another biform variable in the current S (when a subsuming product is deleted, a variable which was biform may become monoform) and go to Step (2). If all biform variables are tried, go to Step (4).

(4) The procedure terminates and all the product in S are desired prime implicants.

Example 3.1.4-2 - Given f = $x_1x_2x_4 \vee x_1x_3 \vee x_2\bar{x}_3 \vee x_2x_4 \vee \bar{x}_3\bar{x}_4$

Following (1), we found that x_3 and x_4 are biform variables. Let us choose x_3.

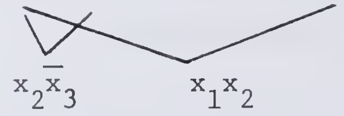
Following (2), we get consensus x_1x_2 for pair, x_1x_3 and $x_2\bar{x}_3$; and con sensus $x_1\bar{x}_4$ for pair, $x_1\bar{x}_3$ and $\bar{x}_3\bar{x}_4$.



Following (3), we choose the remaining biform variable x_4, go back to (2).

Following (2), we generate two consensi as follows:

S: x_1x_3, $x_2\bar{x}_3$, x_2x_4, $\bar{x}_3\bar{x}_4$, x_1x_2, $x_1\bar{x}_4$



But when we add x_1x_2 and $x_2\bar{x}_3$ to S, all of them subsume the products contained in S. So they are eliminated. Now, the current S contains all prime implicants: x_1x_3, x_2x_4, $\bar{x}_3\bar{x}_4$, $x_1\bar{x}_4$, $x_2\bar{x}_3$ and x_1x_2.

Procedure II (Tison method for the derivation of all irredundant disjunctive forms): Assume we have all prime implicants of f, i.e., P, Q, ...,T found in Procedure I. Let S denote the set of these prime implicants. We want to find all irredundant disjunctive forms of f.

(1) Label each prime implicant P with an index number I, denoting as PI.

Using Procedure I with the following modification, let us find which prime implicants generate which prime implicant as consensus.

(2) Choose the first biform variable, say x. We will find consensi with respect to x as follows. If P_1I_1 and P_2I_2 generate a consensus with respect to x, in other words, if $P_1 = xp_1$ and $P_2 = \bar{x}p_2$, then the consensus p_1p_2 is indexed with the combination of the index numbers I_1I_2 as:

$$p_1p_2 \quad I_1I_2.$$

Find all such indexed consensi with respect to x which can be generated from all the prime implicants in S.

(3) Compare each of the indexed products derived in the previous step with all those in S, by regarding each indexed product to be a product consisting

of the original literals and index numbers as new literals (e.g., indexed product $x_2x_3x_4$ 48 is regarded as a product consisting of literals x_2, x_3, x_4, 4, and 8). If any product is subsumed by another, discard it and otherwise add it to S.

(4) Choose next biform variable and return to Step (2). If all the biform variables are tried (each biform variable needs to be tried only once), the indexed products in S denote all consensus relations.

(5) For each prime implicant of f (although non-prime implicants of f are usually contained in the last S, we ignore all these non-prime implicants), find all the indexed products which contain this prime implicant. Then make the disjunction of all the index numbers contained in them.

(i) Then make the conjunction of all these disjunctions. This conjunction is called the <u>Tison function</u>.

(ii) Multiplying out, treating index numbers as switching variables, obtain the complete sum.

(iii) Corresponding to the index numbers in each term of the complete sum, make the disjunction of the prime implicants which have these index numbers. All disjunctive forms obtained are all irredundant disjunctive forms.

<u>Example 3.1.4-3</u> - For the function in Example 3.1.4-2, the prime implicants are x_1x_3, x_2x_4, $\bar{x}_3\bar{x}_4$, $x_1\bar{x}_4$, $x_2\bar{x}_3$, x_1x_2. Let us follow Procedure II to derive all irredundant disjunctive forms.

Step (1): $S = \{x_1x_3 1,\ x_2x_4 2,\ \bar{x}_3\bar{x}_4 3,\ x_1\bar{x}_4 4,\ x_2\bar{x}_3 5,\ x_1x_2 6\}$

Step (2): x_3 and x_4 are biform variables. Let us choose x_3. Two consensi are then generated: $x_1\bar{x}_4 13$, $x_1x_2 15$.

Step (3): The consensi generated in step (2) do not subsume any other product. So the current S = $\{x_1x_31, x_2x_42, \bar{x}_3\bar{x}_43, x_1\bar{x}_44, x_2\bar{x}_35, x_1x_26, x_1\bar{x}_413,$ $x_1x_215\}$.

Step (4): Choose the biform variable x_4 and return to step (2).

Step (2): Three consensi $x_2\bar{x}_323$, x_1x_224 and x_1x_2123 are generated.

Step (3): They do not subsume any other product, so S = $\{x_1x_31, x_2x_42, \bar{x}_3\bar{x}_43,$ $x_1\bar{x}_44, x_2\bar{x}_35, x_1x_26, x_1\bar{x}_413, x_2\bar{x}_323, x_1x_224, x_1x_2123\}$.

Step (4): Since all biform variables have been tried, S contains all consensus relations.

Step (5): Form the Tison function:

$T = (1)\ (2)\ (3)\ (4 \vee 13)\ (5 \vee 23)\ (6 \vee 15 \vee 24 \vee 123)$

Multiplying out, we get $T = 123\ (456 \vee 145 \vee 245 \vee 234 \vee 135 \vee 123)$

$= 123$

Therefore the minimum sum is $x_1x_3 \vee x_2x_4 \vee \bar{x}_3\bar{x}_4$

Besides the Tison's method, there are many other ways to find the minimum sums. A software package named MINIPACK (MINImization PACKage) is under development by R. Cutler. In this package, Tison's method, the branch-and-bound method, the hybrid method (the combination of Tison's method and the branch-and-bound method) and Quine-McCluskey's method [26] with the branch-and-bound method (developed by I. Suwa) are all included. The method used in NETTRA system is a straight-forward version of Tison's method.

3.1.5 Gimpel's Algorithm

Gimpel's algorithm is a method for finding optimal (minimum number of gates is the only cost criterion) three-level NAND networks for any given switching function with only uncomplemented external variables as inputs[*]. Since the problem of designing optimal three-level NOR networks for the given function is equivalent to the problem of designing optimal three-level NAND networks for the dual of the given function, Gimpel's algorithm is also used to design initial networks in the NETTRA system.

The detailed algorithm is very long and complicated [7]. Only the basic ideas are reviewed in this section.

Let us call the output gate of a network the first-level gate. For the three-level NAND network shown in Fig. 3.1.5-1, the function realized at the output of each input gate (third-level gate) is of the form $\overline{T}$ where T is the product of uncomplemented external variables. The conjunction of functions realized at the inputs of each second-level gate has the form

$T_0 \overline{T}_1 \ldots \overline{T}_m$; here again, T_0, T_1, ... T_m for $m \geq 0$, are products of uncomplemented external variables. The function realized by the network in Fig. 3.1.5-1 can thus be expressed as disjunction of these $T_0\overline{T}_1 \ldots \overline{T}_m$ expressions. Gimpel's algorithm aims at finding appropriate products T_1, ..., T_m (each corresponds to a third-level gate) and expressions $T_0\overline{T}_1 \ldots \overline{T}_m$ (each corresponds to a second-level gate) so that the total number of gates needed is minimum. The following definitions and examples are useful for our discussions.

Definition 3.1.5-1 - A frontal term is a product of different uncomplemented variables or the constant 1.

Example 3.1.5-1 - x, wxyz, 1 and wy are frontal terms whereas $x\overline{y}$ and $xy \vee w$ are not. Every input gate in Fig. 3.1.5-1 realizes a function of the

[*] In [7], this is called a TANT network. (Three-level AND-NOT network with True inputs.)

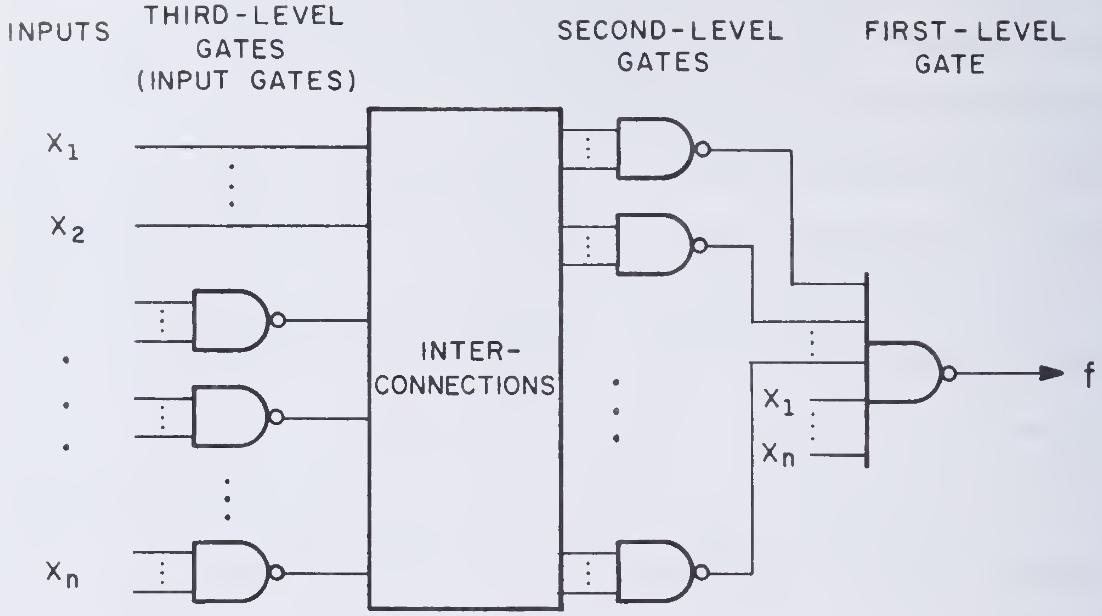


Fig. 3.1.5-1 A three-level TANT network

form $\overline{P}$ where P is a frontal term.

Definition 3.1.5-2 - *A permissible expression** is any switching expression (not identically 0) of the form $T_0\overline{T}_1 \ldots \overline{T}_m$ for $m \geq 0$ where each T_i is a frontal term. In a permissible expression $T_0\overline{T}_1 \ldots \overline{T}_m$, T_0 is called the *head* of the expression, whereas $\overline{T}_1 \ldots \overline{T}_m$ is called the *tail* of the expression and each T_i is called a *tail factor* for $i = 1, \ldots, m$.

Example 3.1.5-2 - $xy(\overline{wx})(\overline{yz})$, $xy\overline{z}$, x, 1, $\overline{v}(\overline{wxyz})$ and $\overline{y}x\overline{z}w$ are permissible expressions with heads xy, xy, x, 1, 1 and xw, respectively, whereas xvy, $xy(\overline{xy})$ and $x(\overline{1})$ are not permissible expressions.

Definition 3.1.5-3 - An *irredundant permissible expression* is a permissible expression in which no factor (either head factor or tail factor) can be removed without changing the function expressed.

Definition 3.1.5-4 - A *TANT expression* is an expression of the form $E_1 \vee E_2 \vee \cdots \vee E_n$ where E_i is a permissible expression.

Example 3.1.5-3 - $w(\overline{wxy}) \vee xy(\overline{wxy})$ is a TANT expression. The TANT expression corresponds to the output of the TANT network shown in Fig. 3.1.5-1.

Definition 3.1.5-5 - A *permissible implicant* of a switching function f is a function which implies f and which can be written as a permissible expression.

Example 3.1.5-4 - Given $f(w,x,y) = \overline{y}w \vee w\overline{x} \vee y\overline{w}x$. The irredundant permissible expressions which imply f are listed below:

$$\overline{w}xy,\ (\overline{wx})xy,\ (\overline{wy})xy,\ (\overline{wxy})xy$$

*In a TANT network, the conjunction of functions realized at inputs of each second-level gate is a permissible expression.

$w(\overline{xy}),\ w(\overline{wxy})$

$w\overline{x},\ w(\overline{wx})$

$w\overline{y},\ w(\overline{wy})$

$wx\overline{y},\ wx(\overline{wy}),\ wx(\overline{xy}),\ wx(\overline{wxy})$

$w\overline{x}y,\ w(\overline{wx})y,\ w(\overline{xy})y,\ w(\overline{wxy})y$

$w\overline{x}\overline{y},\ w(\overline{wx})\overline{y},\ w\overline{x}(\overline{wy}),\ w(\overline{wx})(\overline{wy})$

These 22 irredundant permissible expressions are classified into 7 groups. Each member in the same group expresses the same permissible implicant. The members in the first group, $(\overline{wx})xy$, $(\overline{wy})xy$ and $(\overline{wxy})xy$ express the same permissible implicant $\overline{w}xy$. As a matter of fact, the former three can be obtained by augmenting the tail factor with head variables in the latter expression $\overline{w}xy$ in three different ways: $\overline{w}xy = \overline{w}xy \vee \overline{x}xy = (\overline{w} \vee \overline{x})xy = (\overline{wx})xy$; $\overline{w}xy = \overline{w}xy \vee \overline{x}xy \vee \overline{y}xy = (\overline{w} \vee \overline{x} \vee \overline{y})xy = (\overline{wxy})xy$; and $\overline{w}xy = \overline{w}xy \vee \overline{y}xy = (\overline{w} \vee \overline{y})xy = (\overline{wy})xy$.

Definition 3.1.5-6 - The permissible expression which cannot be obtained by augmenting the tail factor with head variables in any other permissible expression, such as $\overline{w}xy$ in Example 3.1.5-4, is called the principal expression for a given permissible implicant. Its tail and tail factors are called the principal tail and principal tail factors, respectively.

Definition 3.1.5-7 - A prime permissible implicant (or PP-implicant) of a function f is a permissible implicant such that if any principal tail factor is removed from its principal expression, the resulting function will not be a permissible implicant of f.

Definition 3.1.5-8 - A lower PP-implicant is a PP-implicant properly implying a prime implicant. An upper PP-implicant is a PP-implicant which is not lower.

Definition 3.1.5-9 - A permissible implicant is said to be simple if all its principal tail factors are complemented variables (in other words, if it can be expressed as a product of literals); otherwise, the permissible implicant is said to be compound.

In the following, theorems are stated without proofs.

Theorem 3.1.5-1 - If a permissible implicant P is compound, then P is an upper PP-implicant.

Theorem 3.1.5-2 - Every prime implicant is an upper PP-implicant.

Example 3.1.5-5 - Consider the function f given in Example 3.1.5-4. There are six PP-implicants[*] for f, as shown in Table 3.1.5-1. Among them, there is only one compound PP-implicant, there are three prime implicants and there are four upper PP-implicants. Please notice that each prime implicant is an upper PP-implicant, since it does not properly imply itself.

Table 3.1.5-1 The PP-implicants of the function in Example 3.1.5-4.

PP-Implicant	Compound	Prime Implicant	Upper PP-Implicant
$w(\overline{xy})$	x		x
$xy\overline{w}$		x	x
$w\overline{x}$		x	x
$w\overline{y}$		x	x
$wx\overline{y}$			
$wy\overline{x}$			

Theorem 3.1.5-3 - Let the minimization of the number of gates be the first cost criterion and let the minimization of the number of connections

[*] A way to find all PP-implicants will be explained later.

be the second cost criterion. The permissible implicants in each minimum TANT expression are PP-implicants.

Theorem 3.1.5-4 - Let the minimization of the number of gates be the cost criterion. For any function f, there exists a minimum TANT expression in which every permissible implicant is an upper PP-implicant.

Definition 3.1.5-10 - The maximum permissible implicant with head H is the disjunction of all permissible implicants with the common head H.

Theorem 3.1.5-5 - The maximum permissible implicant P_0 with head H of a function f can be written as

$$P_0 = H\ (\pi_1 \vee \pi_2 \vee \cdots \vee \pi_m)$$

where $\{\pi_1, \pi_2 \ldots \pi_m\}$ is the set of prime implicants which is implied by the minterms with head H.

Definition 3.1.5-11 - Let $\{S_1, \ldots, S_m\}$ and $\{T_1, \ldots, T_n\}$ be sets of frontal terms. The set $\{S_i\}$ is said to be an overlay for $\{T_i\}$ if for each T_i there exists some some S_j such that $S_j \supseteq T_i$. If no S_i can be removed without destroying this property, then $\{S_i\}$ is said to be an irredundant overlay.

Theorem 3.1.5-6 - Let $H\overline{T}_1 \ldots \overline{T}_n$ be the principal expression for the maximum permissible implicant with head H of a function f. Then $H\overline{S}_1 \ldots \overline{S}_m$ is the principal expression for a PP-implicant of f if and only if $\{S_1, \ldots, S_m\}$ is an irredundant overlay (except the unit overlay {1}) for $\{T_1, \ldots, T_n\}$.

Example 3.1.5-6 - The function given in Example 3.1.5-4 has three prime implicants of $w\overline{y}$, $w\overline{x}$ and $\overline{w}xy$. Minterm $w\overline{x}\overline{y}$ implies prime implicants $w\overline{x}$ and $w\overline{y}$. Thus by Theorem 3.1.5-5, $w(w\overline{y} \vee w\overline{x}) = w\ (\overline{xy})$ yields the maximum permissible implicant with head w. Minterm $wx\overline{y}$ implies prime implicants of $w\overline{y}$ only. Thus we obtain $wx(w\overline{y}) = wx\overline{y}$ which is the maximum permissible implicant with head wx. For the similar reason $\overline{w}xy$ is a maximum permissible implicant wy.

There is only one minterm $\bar{w}xy$ with head xy, so $\bar{w}xy$ is also a maximum permissible implicant.

Take the maximum permissible implicant $w(\overline{xy})$. There are three irredundant overlays (except $\{1\}$) for $\{xy\}$: $\{xy\}$, $\{x\}$ and $\{y\}$. So we have three PP-implicants with head w: $w(\overline{xy})$, $w\bar{x}$, $w\bar{y}$. Take the maximum permissible implicant $\bar{w}xy$. There is only one irredundant overlay except $\{1\}$ for $\{w\}$: $\{w\}$. So we have only one PP-implicant $\bar{w}xy$ with head xy.

Applying Theorem 3.1.5-5 and Theorem 3.1.5-6, we can generate all PP-implicants for any function f and from these PP-implicants we can find all upper PP-implicants. But not all PP-implicants or upper PP-implicants are necessary for finding minimum TANT expressions. We give the following definitions, theorems and examples to explain how to get minimum TANT expressions.

Definition 3.1.5-12 - A permissible implicant P_0 is dominant if for every permissible implicant P, either $P_0P = 0$ or $P_0 \supseteq P$.

Definition 3.1.5-13 - A permissible implicant is quasi-simple if at most one of its principal tail factors is compound.

Definition 3.1.5-14 - A major PP-implicant is any upper PP-implicant not properly implying a dominant quasi-simple PP-implicant.

Theorem 3.1.5-7 - A maximum permissible implicant with head H of a function f is dominant if and only if there exists at least one prime implicant with head H, and the disjunction of all the prime implicants with head H is disjoint with the disjunction of all the remaining prime implicants.

Theorem 3.1.5-8 - For any function f there exists a minimum TANT expression such that each permissible implicant is a major PP-implicant.

Figure 3.1.5-7 - For the previous example, there are two major PP-implicants: $w(\overline{xy})$ and $\bar{w}xy$. The only question that remains is which particular irredundant permissible expressions should be chosen for each of these two major

PP-implicants. There are four irredundant permissible expressions for $\overline{w}xy$: $\overline{w}xy$, $(\overline{wx})xy$, $(\overline{wy})xy$ and $(\overline{wxy})xy$ and there are two for $w(\overline{xy})$: $w(\overline{xy})$ and $w(\overline{wxy})$. If we choose the last two expressions in both cases we are able to share input gates. The resulting minimum TANT network is shown in Fig. 3.1.5-2.

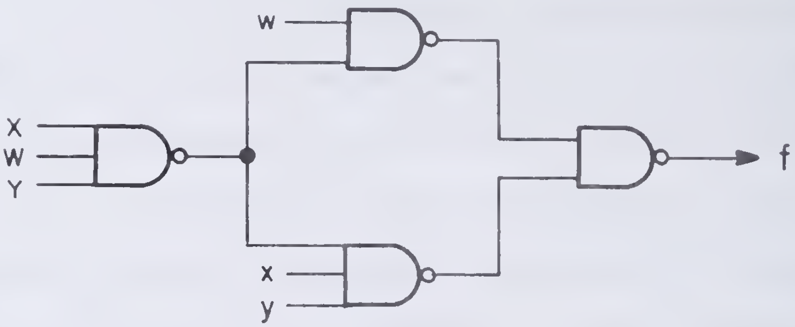


Fig. 3.1.5-2 The minimum TANT network for f.

The selection of major PP-implicants for a minimum TANT expression is essentially a minimum covering problem. The concepts of major PP-implicant table and cc-table are introduced in [7] to try to solve the minimum covering problem, with details omitted here. A procedure to speed up the processing of the cc-table which is the most time-consuming part of Gimpel's algorithm [7] is discussed in [10].

Also notice that Gimpel's algorithm based on upper PP-implicants yields only networks with a minimal number of gates, not considering the number of connections, so a network which has the minimal number of gates as the primary objective and the minimal number of connections as the secondary objective may not be obtained.

3.1.6 Level-restricted initial network method

The level-restricted initial network method can expand a two-level or a three-level network, which is based on Tison's method, into a level-restricted network when both the fan-in/fan-out restrictions and the level-restriction are imposed. For the sake of convenience, we will assume that both uncomplemented and complemented external variables are permitted as inputs in the following discussions.

Assume that a minimum product for the given function f consists of ℓ alterms, and the i-th alterm ($1 \leq i \leq \ell$) consists of n_i literals. The corresponding two-level network is shown in Fig. 3.1.6-1. Let the maximum fan-in of a gate be FI, the maximum fan-out of an external variable be FOX and the maximum fan-out of a gate be FO. In Fig. 3.1.6-1, if ℓ > FI, then there is no fan-in problem* at the first-level gate G, and we need to check the fan-in problems of the higher level gates only. If ℓ > FI, then we can solve this fan-in problem using the following approach:

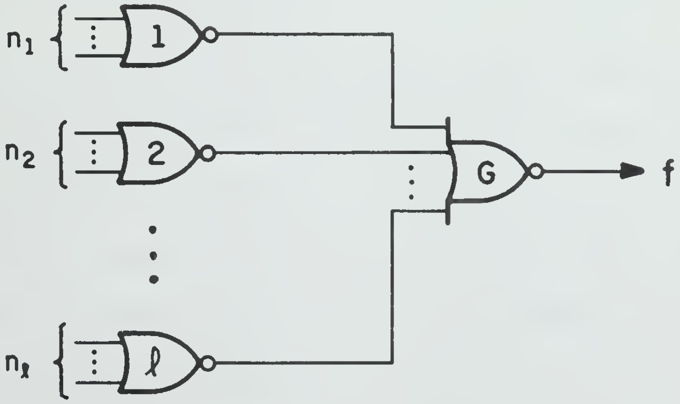


Fig. 3.1.6-1 Two-level network based on the minimum product

[*] In this paper, whenever we mention that "there is a fan-in problem at gate G", we mean that the number of fan-in of gate G is greater than the restriction FI. The fan-out problem at a gate or an external variable is similarly defined.

(1) Partition the input functions of gate G into at most FI groups.

(2) Realize the disjunction of the functions of each group by a two-level subnetwork.

(1) solves the fan-in problem at gate G. (2) increases the number of levels by one and may generate some fan-in/fan-out problems in those two-level subnetworks. But the fan-in problems can be solved by applying procedures (1) and (2) repeatedly. It is easy to see that networks derived by this approach will have a tree structure, hence there is no fan-out problems for gates. There may be fan-out problems for external variables, but they can be solved by adding extra inverters.

Following the above procedures, we can always get a level-restricted network; since each time these procedures are applied, the number of levels of the network is increased at most by 1* (when both uncomplemented and complemented external variables are permitted as inputs, as we assumed). We can also follow these procedures repeatedly until a fan-in/fan-out restricted network is obtained.

In procedure (1), how to divide the ℓ input functions into FI groups is an important problem. An improper formation of groups may generate more fan-in/fan-out problems in higher-level gates. Consider the network shown in Fig. 3.1.6-2(a). Suppose the disjunction of functions f_1, f_2, ..., and f_k is to be realized by a two-level single-output subnetwork, where f_i is the output function of gate i fed by n_i inputs (external variables). In the worst case, all inputs (external variables) of the gates for f_1, f_2, ..., and f_k are different. Assume they are x_1, x_2, ..., $x_{n_1 + n_2 + \cdots + n_k}$ (for simplicity, assume

*In the case that only uncomplemented external variables are available, the number of levels of the network is increased at most by two each time the above procedures are applied. In this case it is also possible that the number of levels does not increase after applying procedures.

all uncomplemented). Then

$$\bigvee_{i=1}^{k} f_i = \bar{x}_1 \cdots \bar{x}_{n_1} \vee \bar{x}_{n_1+1} \cdots \bar{x}_{n_1+n_2} \vee \cdots$$

$$x_{n_1+n_2+\dots+n_{k-1}+1} \cdots \bar{x}_{n_1+n_2+\dots n_k}$$

or

$$\overline{\bigvee_{i=1}^{k} f_i} = (x_i \vee x_2 \vee \cdots \vee x_{n_1})(x_{n_1+1} \vee \cdots \vee x_{n_1+n_2}) \cdots$$

$$(x_{n_1+n_2+\dots+n_{k-1}+1} \vee \cdots \vee x_{n_1+n_2+\dots+n_k}) \qquad (3.1.6\text{-}1)$$

If we multiply out equation (3.1.6-1), we can get a disjuntive form which has $\prod_{i=1}^{k} n_i$ terms and each term is a product of k literals. Taking the complement of this disjunctive form, we can get a conjunctive form for $\bigvee_{i=1}^{k} f_i$. This conjunctive form has $\prod_{i=1}^{k} n_i$ alterms and each alterm is a disjunction of k literals. A two-level NOR subnetwork can be obtained based on this conjunctive form. The total number of NOR gates in this subnetwork is $\prod_{i=1}^{k} n_i + 1$, i.e., $\prod_{i=1}^{k} n_i$ gates in the higher level and another one in the output level, See Fig. 3.1.6-2(b). An example is given gelow:

<u>Example 3.1.6-1</u> - Suppose $f_1 = \bar{x}_1\bar{x}_2$, $f_2 = \bar{x}_3\bar{x}_4$, i.e., $n_1 = 2$ and $n_2 = 2$ are realized as part of a network, as shown in Fig. 3.1.6-3. Then

$$\overline{f_1 \vee f_2} = (x_1 \vee x_2)\ (x_3 \vee x_4)$$

$$= x_1x_3 \vee x_1x_4 \vee x_2x_3 \vee x_2x_4$$

$$= \overline{(\bar{x}_1 \vee \bar{x}_3)(\bar{x}_1 \vee \bar{x}_4)(\bar{x}_2 \vee \bar{x}_3)(\bar{x}_2 \vee \bar{x}_4)}$$

So a two-level subnetwork can be obtained in Fig. 3.1.6-3(b)

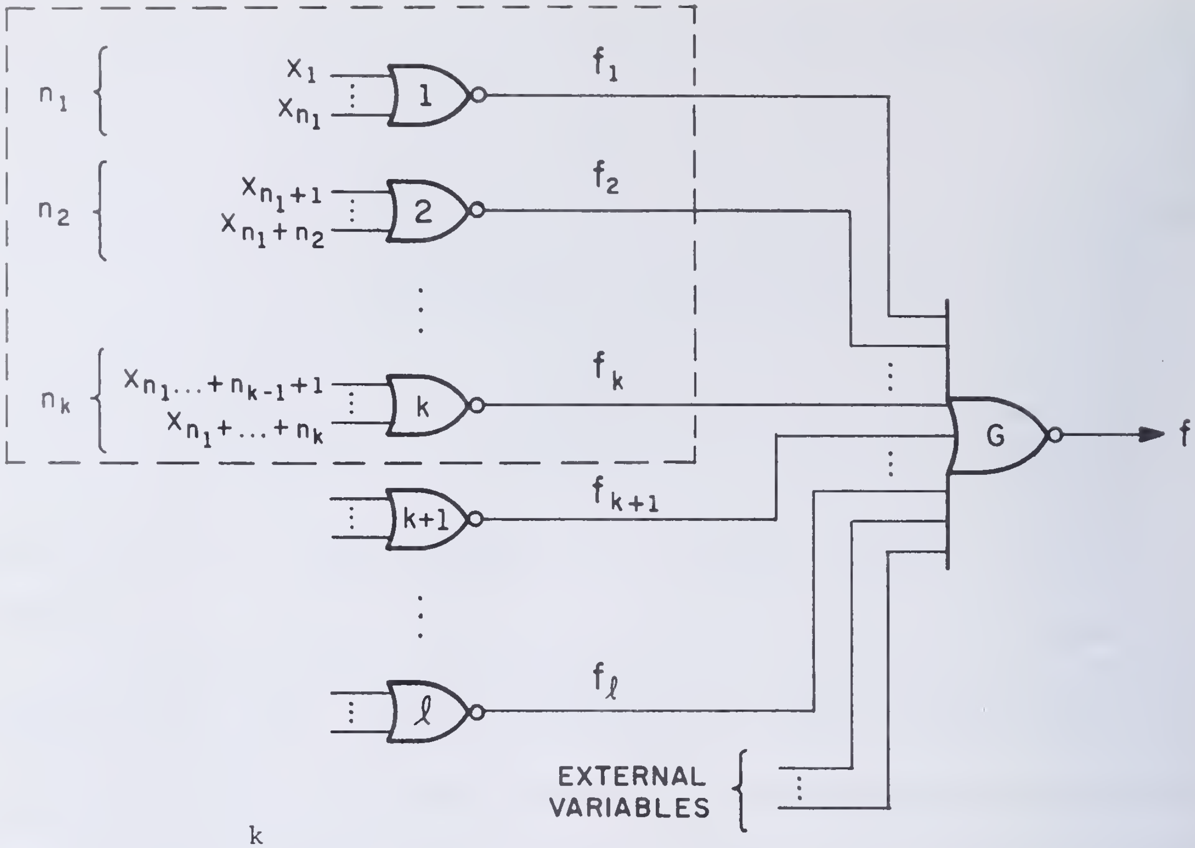


Fig. 3.1.6-2(a) $\bigvee_{i=1}^{k} f_i$ is to be realized by a two-level single-output subnetwork

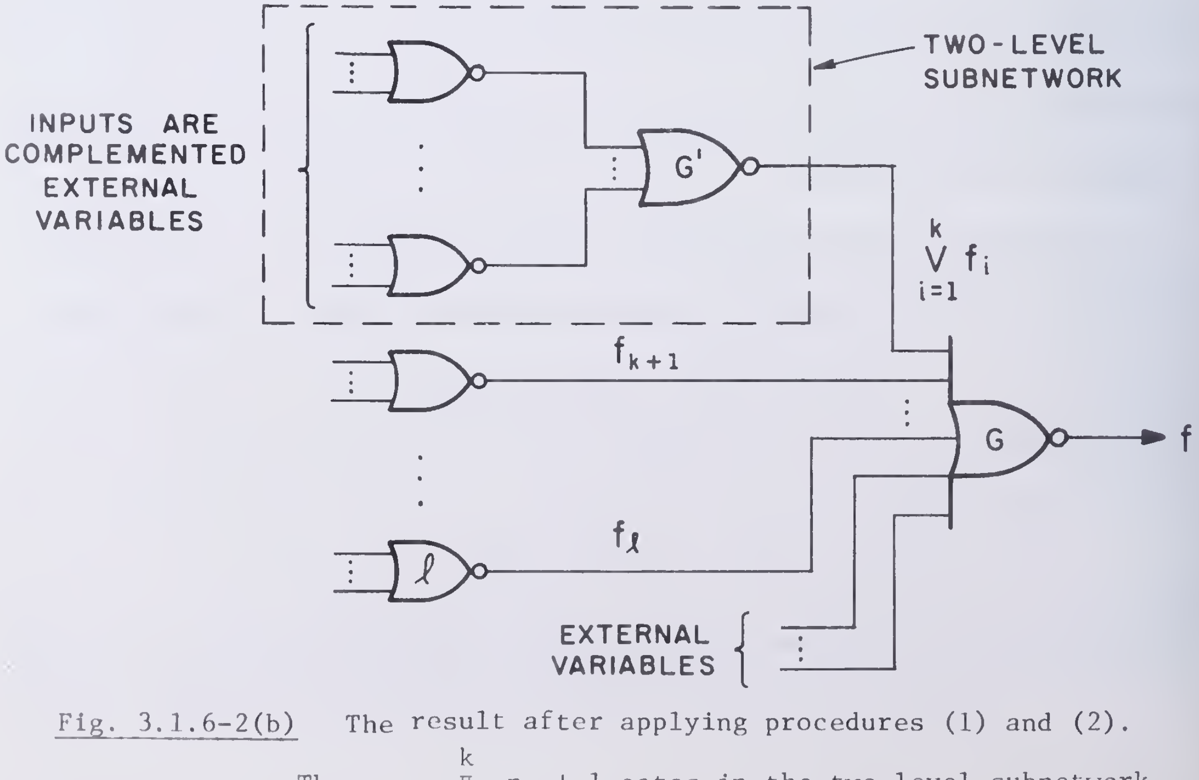
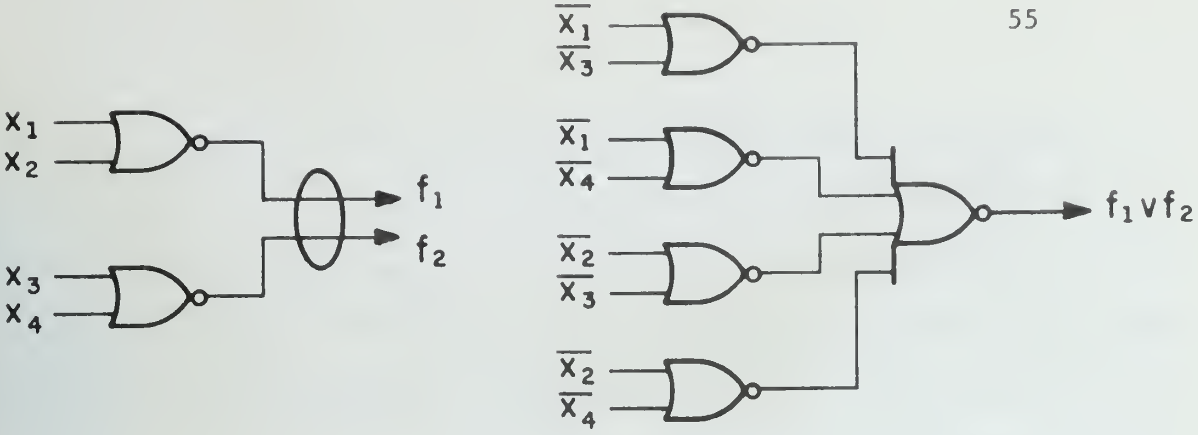


Fig. 3.1.6-2(b) The result after applying procedures (1) and (2). There are $\prod_{i=1}^{k} n_i + 1$ gates in the two-level subnetwork.



(a) The original network

(b) The network after applying procedures (1) and (2)

Fig. 3.1.6-3 Example 3.1.6-1

Of course, the above analysis is the worst case. Usually some x_i and $\bar{x}_i$ both appear in equation (3.1.6-1) so that many terms become identically zero after multiplying out equation (3.1.6-1) - this means that the total number of the second-level gates or the total number of fan-in of gate G' in Fig. 3.1.6-2 is less than $\prod_{i=1}^{k} n_i$. Sometimes x_i (or $\bar{x}_i$) appears in more than one alterm in equation (3.1.6-1) so that many terms may contain fewer literals than k and also some terms subsume other terms and hence can be eliminated after multiplying out equation (3.1.6-1). In general, if more pairs of x_i and $\bar{x}_i$ appear in different alterms and x_i or $\bar{x}_i$ appears in more alterms, then fewer gates will have fan-in problems and fewer external variables will have fan-out problems in the resulting two-level subnetwork.

Example 3.1.6-2 - Consider the two-level network in Fig. 3.1.6-4(a), where $f_1 = x_1x_2\bar{x}_3$, $f_2 = \bar{x}_1\bar{x}_2\bar{x}_3$, $f_3 = x_1x_4x_5$ and FI = FOX = 2. Since the output gate has 3 inputs, let us first solve this fan-in problem. It is obvious that if we realize the disjunction of any two of the three functions f_1, f_2 and f_3 by a two-level subnetwork, we can reduce the fan-in of the output gate by 1. Fig. 3.1.6-4(b) through Fig. 3.1.6-4(d) show the results of three possible ways of partitioning f_1, f_2 and f_3 into two groups. In Fig. 3.1.6-4(b), $f_1 \vee f_2$ is realized. $f_1 \vee f_2$ has two pairs of complemented and uncomplemented external variables and $\bar{x}_3$ appears twice. In Fig. 3.1.6-4(c), $f_1 \vee f_3$ is realized. $f_1 \vee f_3$ has no complemented and uncomplemented pairs but x_1 appears twice. In Fig. 3.1.6-4(d), $f_2 \vee f_3$ is realized. $f_2 \vee f_3$ has one complemented and uncomplemented pair but no literal appears two or more times. Obviously, the result obtained in Fig. 3.1.6-4(b) is the best among three. Since this network still has fan-in problems, we apply the similar procedures to solve these problems. The fan-in/fan-out restricted network which results is shown in Fig. 3.1.6-5.

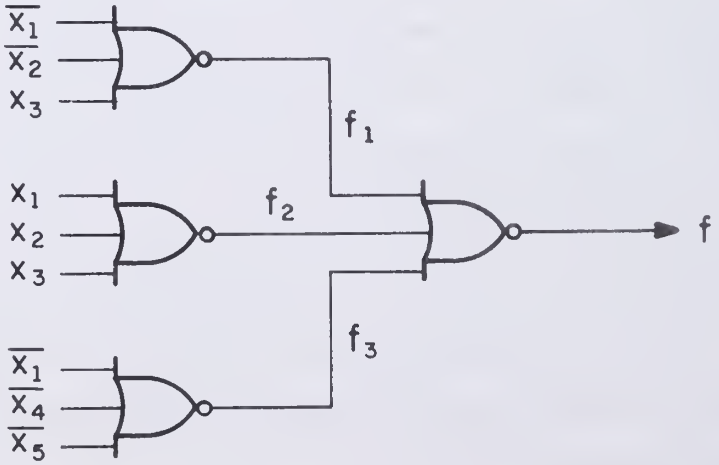


Fig. 3.1.6-4(a) The original two-level network

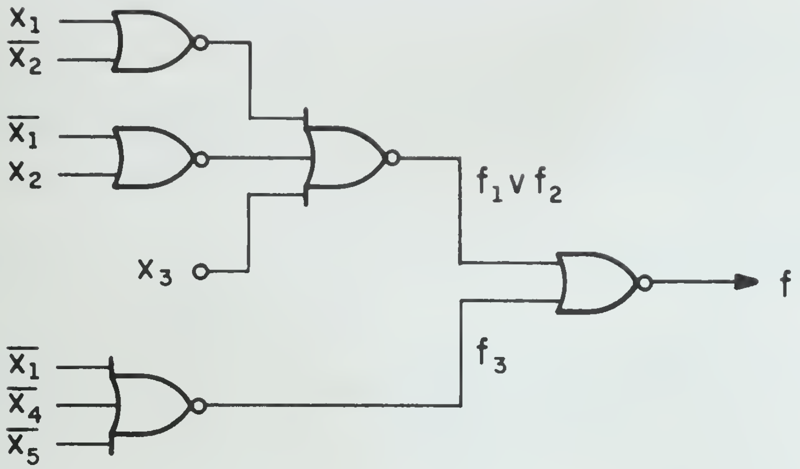


Fig. 3.1.6-4(b) The network after realizing f_1 V f_2

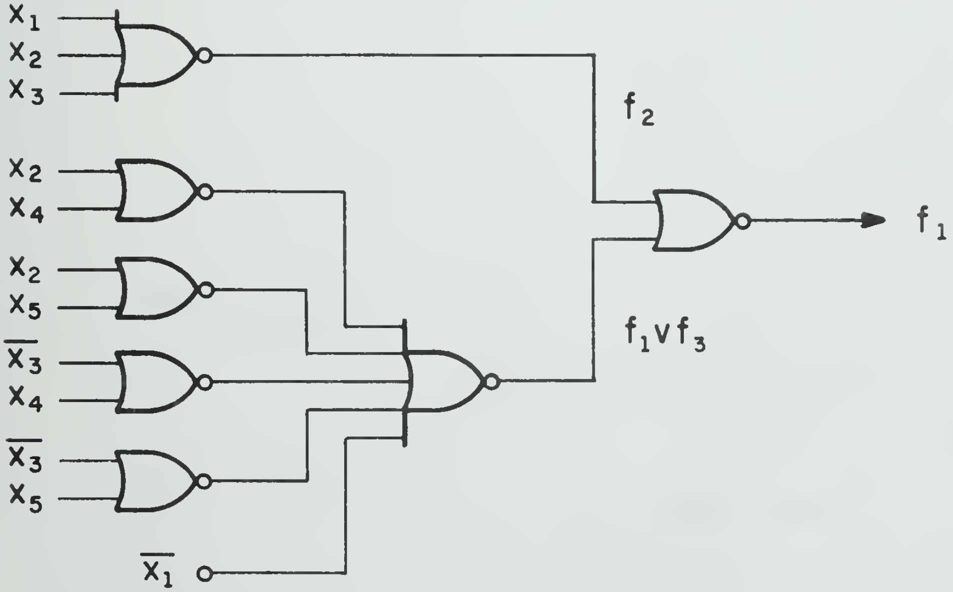


Fig. 3.1.6-4(c) The network after realizing f_1Vf_3

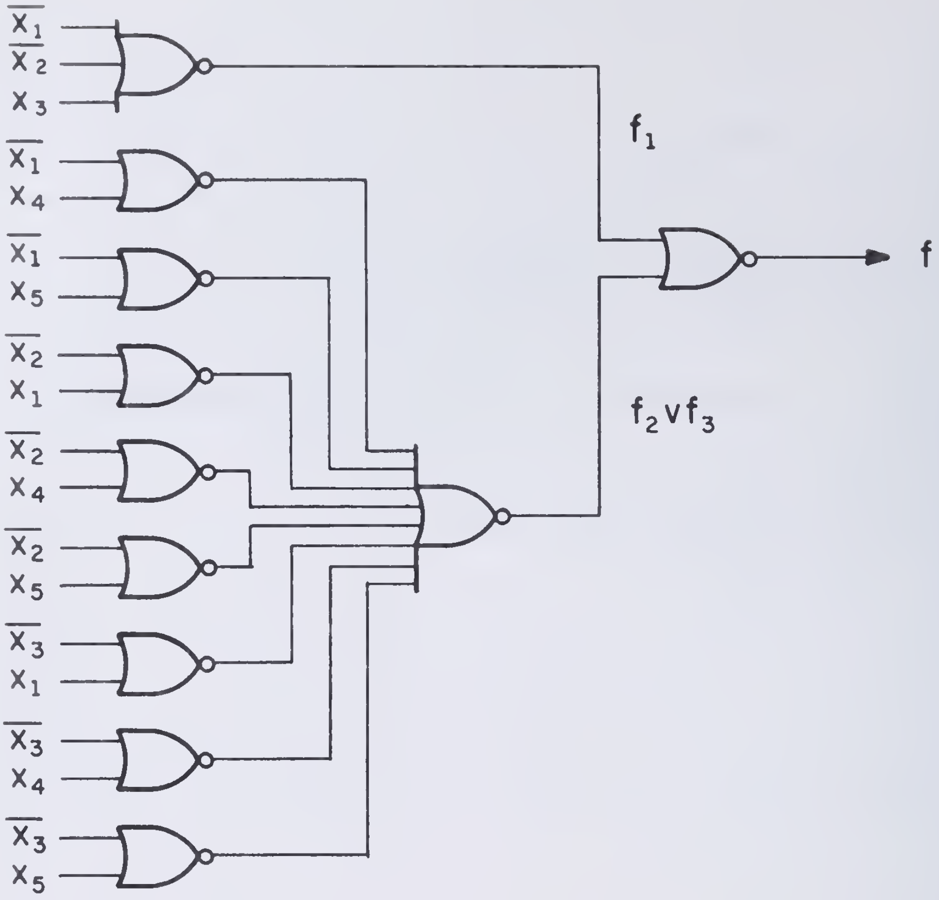


Fig. 3.1.6-4(d) The network after realizing f_2Vf_3

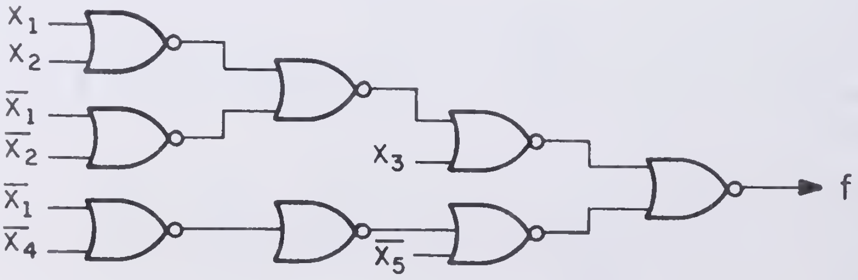


Fig. 3.1.6-5 The fan-in/fan-out restricted network for f in Example 3.1.6-2.

In general if there are more common literals and if there are more pairs of complemented and uncomplemented literals appearing in equation (3.1.6-1), then the resultant two-level subnetwork will have fewer number of gates and also fewer gates will have fan-in problems.

The following two criteria (1 as the primary criterion and 2 as the secondary criterion) are used in partitioning the input functions of a gate into groups to solve the fan-in problems:

<u>Criterion 1</u>: Select the group which has the largest number of common literals.

<u>Criterion 2</u>: Select the group which has the largest number of pairs of complemented and uncomplemented external variables.

The implementation of procedures (1) and (2) and the way to treat multiple-output network problems are detailed in [12].

3.2 <u>Fan-in/fan-out restricted transformations</u>

In this section we discuss the fan-in/fan-out restricted transformation method [24]. Any NOR network that does not satisfy the given fan-in/fan-out restrictions can be transformed by this method into a fan-in/fan-out restricted network. A set of six transformations (T_1 through T_6) will be reviewed one by one. The following definitions are introduced first:

Let external variables be $x_1, x_2, \ldots, x_n$ (and $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n$ when uncomplemented external variables are available) and the network has R gates $g_1, g_2, \ldots, g_R$. Let the maximum fan-in of a gate, the maximum fan-out of a gate (not an output gate), the maximum fan-out of an external variable and the maximum fan-out of an output gate be FI, FO, FOX and FOO, respectively.

<u>Definition 3.2-1</u> - A gate g_j is said to be an <u>immediate successor</u> of gate g_i (or external variable x_i) if and only if the output of g_i (or external variable x_i) is connected to g_i as an input. Let IS(g_i) (or IS(x_i)) denote

the set of all immediate successors of the gate g_i (or external variable x_i).

Definition 3.2-2 - A gate g_i is said to be an immediate predecessor of gate g_i if and only if $g_i \varepsilon IS(g_j)$. Let $IP(g_i)$ denote the set of all immediate predecessors of the gate g_i.

Definition 3.2-3 - Condition set C1 consists of the following conditions for the application of transformation T1:

(1) $$|IS(g_i)|^* > \begin{cases} FO \text{ if } g_i \text{ is a non-output gate} \\ FOO \text{ if } g_i \text{ is an output gate} \end{cases}$$

(2) $|IP(g_i)|$ must be less than or equal to FI

(3) For every gate $g_k \varepsilon IP(g_i)$, $|IS(g_k)|$ must be less than FO (or FOO for output gates). For every external variable $x_k \varepsilon IP(g_i)$, $|IS(x_k)|$ must be less than FOX.

Definition 3.2-4 - Condition set C2 consists of the following conditions for the application of transformation T2 which is used to solve the fan-out problem of an external variable:

(1) $|IS(x_j)| > FOX$

(2) There exists a gate g_i whose only immediate predecessor is external variable x_j : i.e., $IP(g_i) = \{x_j\}$.

(3) For this particular gate g_i,

(a) $$|IS(g_i)| < \begin{cases} FO \text{ if } g_i \text{ is a non-output gate} \\ FOO \text{ if } g_i \text{ is an output gate} \end{cases}$$

Definition 3.2-5 - Condition set[†] C4 consists of the following conditions for the application of transformation T4:

(1) $|IP(g_i)| > FI$

[*] $|IS(g_i)|$ and $|IP(g_i)|$ represent the number of elements in sets $IS(g_i)$ and $IP(g_i$ respectively.

[†] Condition sets which are numbered in correspondence to the transformation numbe by Legge []3] are used here, so there are no C_3 and C_5.

(2) There exists a gate g_j such that:

(a) $$|IS(g_j)| < \begin{cases} FO \text{ if } g_j \text{ is a non-output gate} \\ FOO \text{ if } g_j \text{ is an output gate} \end{cases}$$

(b) every immediate predecessor of g_j is also an immediate predecessor of g_i.

(3) $|IP(g_i)| - |IP(g_j)| < FI$

Definition 3.2-6 - Condition set C6 consists of the following conditions for the application of transformation T6:

(1) There exists a set of gates and/or external variables $K = \{x_{i_1}, \ldots, x_{i_t}, g_{i_{t+1}}, \ldots, g_{i_k}\}$ (t may be 0 or t may equal k in some cases) and a set of gates $H = \{g_{j_1}, g_{j_2}, \ldots, g_{j_h}\}$ such that for every $g_{j_r} \varepsilon H$, $IP(g_{j_r}) \supseteq K$ holds, for every $g_{i_s} \varepsilon K$, $IS(g_{i_s}) \supseteq H$ holds, and for every $x_{i_r} \varepsilon K$, $IS(x_{i_r}) \supseteq H$ holds.

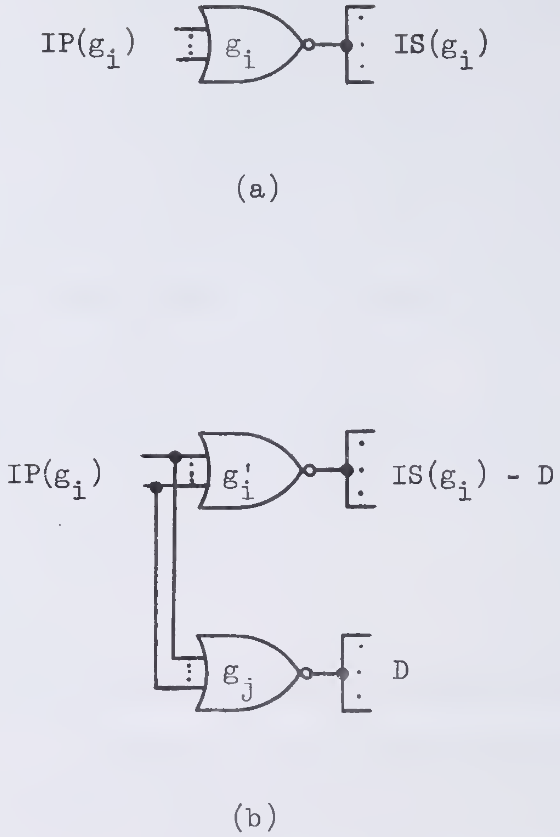
(2) $2 \leq k \leq FI$.

(3) $2 \leq h \leq (FO)^2$

(4) The number of gates in K with fan-out problems plus the number of gates in H with fan-in problems must be at least 2.

Transformations T1, T2 and T3 are used for solving fan-out problems, whereas transformations T4 and T5 are used for solving fan-in problems. Transformation T6 is used to solve the composite problem.

Transformation T1: If a gate (not output gate) g_i, shown in Fig. 3.2-1(a) has a fan-out problem and condition set C1 is satisfied, then transformation T1 is applied as follows: a gate, g_j is added to the network, as shown in Fig. 3.2-1(b) where the original gate g_i is denoted with g'_i after modification. The immediate predecessors of g_i are connected as inputs to g_j. As many output connections as necessary to correct g_i's fan-out problem — up to a maximum number



(a)

(b)

Fig. 3.2-1 (a) Gate g_i with fan-out problem

(b) Gate g_i's fan-out problem is solved by applying T1. After the transformation, the following relations hold. If $|IS(g_i)| \leq 2 \times FO$, then $|IS(g_i) - D| = FO$ and $|D| \leq FO$. If $|IS(g_i)| > 2 \times FO$, then $|IS(g_i) - D| > FO$ and $|D| = FO$. Here D denotes the set of gates fed by the transferred connections.

equal to FO — are transferred from g_i to g_j (let the set of gates fed by transferred connections be designated D). Thus, in the case when $|IS(g_i)|$ is originally greater than two times FO, g_i will still have a fan-out problem (although less than the original problem) following the transformation, and further transformations will have to be employed.

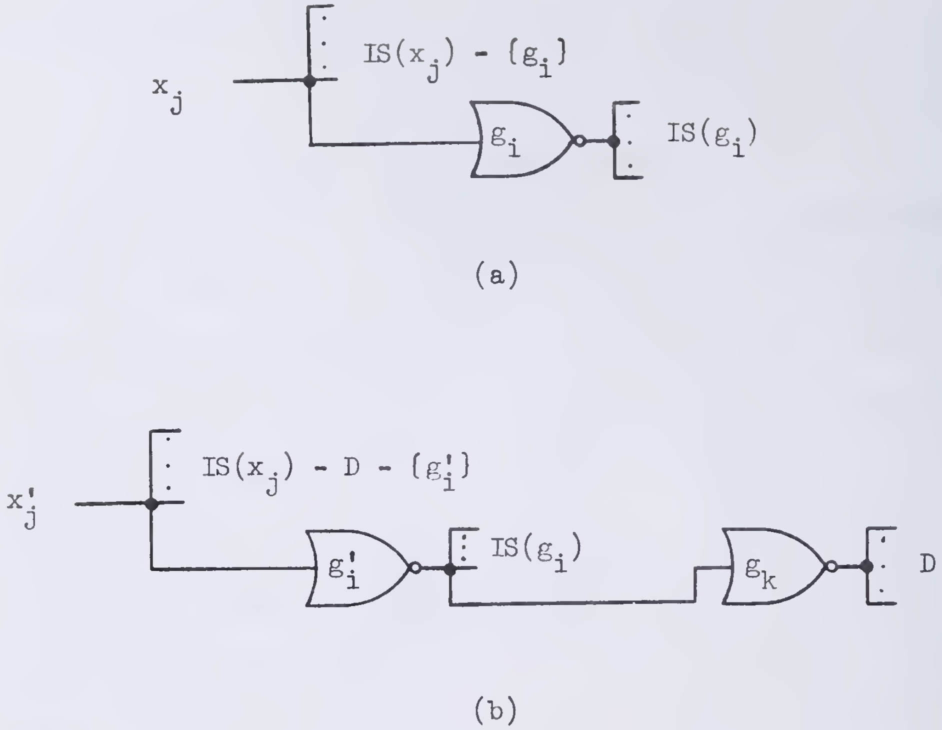
The case for an output gate g_i can be treated similarly.

Transformation T2: For any gate g_i and any external variable x_j satisfying condition set C2, the fan-out problem of x_j is solved as follows: As shown in Fig. 3.2-2, add a new gate, g_k, as an immediate successor to gate g_i and transfer as many output connections (excluding the connection to g_i) as necessary to correct x_j's fan-out problem -- up to a maximum number equal to FO -- from x_j to the outputs of g_k (let the set of gates fed by the transferred connections be designated D). Then, the function x_j is available at the output of gate g_k. Thus, in the case when $|IS(x_j)|$ is originally greater than FO + FOX, x_j will still have a fan-out problem (although less than the original problem) following the transformation, and further transformations will have to be employed.

Transformation T3: This transformation is used to solve the fan-out problem of a gate g_i (or external variable x_i) when neither condition set C1 nor condition set C2 is satisfied.

There are three general cases (each case will be discussed for a non-output gate, g_i, but the transformations for output gates and external variables are similar):

(1) If gate g_i has a fan-out problem and FO < $|IS(g_i)| \leq (2 \times FO) - 1$, then the corresponding transformation results in Fig. 3.2-3. An output from gate g_i is connected to a new gate, g_j. An output from g_j in turn is connected to another new gate, g_k. As many output connections as necessary to correct



<u>Fig. 3.2-2</u> (a) $IS(x_j)$ exceeds the fan-out constraint .

(b) After application of T2, x_j has only $|IS(x_j) - D|$ immediate successors. However, if $|IS(x_j)| \leq FO + FOX$ originally, then $|IS(x_j) - D| = FOX$ and $|D| \leq FO$ after the transformation. If $|IS(x_j)| > FO + FOX$ originally, then $|IS(x_j) - D| > FOX$ and $|D| = FO$ after the transformation.

g_i's fan-out problem are transferred from g_i to g_k (let the set of gates fed by the transferred connectsions be designated D).

(2) If gate g_i has a fan-out problem and $(2 \times FO) - 1 < |IS(g_i)| \leq (FO)^2 + FO - 1$, then the corresponding transformation results in Fig. 3.2-4. The only difference between this transformation and that shown in Fig. 3.2-3 is that gate g_j now fans out to ℓ newly added gates, $g_{k_1}, g_{k_2}, \ldots, g_{k_\ell}$ where ℓ is the smallest integer such that $(\ell + 1) \times FO - 1 \geq |IS(g_i)|$. Output connections are transferred from g_i to $g_{k_1}, g_{k_2}, \ldots, g_{k_\ell}$ such that $g_i, g_{k_1}, g_{k_2}, \ldots, g_{k_{\ell - 1}}$ each will have a fan-out of FO while g_{k_ℓ} will have a fan-out of $|IS(g_i)| - \ell \times FO + 1$ (which will be $\leq$ FO). Let the sets of connections transferred to $g_{k_1}, \ldots, g_{k_\ell}$ be designated by $D_1, \ldots, D_\ell$, respectively.

(3) If gate g_i has a fan-out problem and $|IS(g_i)| > (FO)^2 + FO - 1$ then the corresponding transformation results in Fig. 3.2-5. The only difference between this transformation and that described in Fig. 3.2-4 is that gate g_j now fans out to ℓ newly added gates, $g_{k_1}, g_{k_2}, \ldots, g_{k_\ell}$, where ℓ is equal to FO. Output connections are transferred from g_i to $g_{k_1}, g_{k_2}, \ldots, g_{k_\ell}$, such that $g_j, g_{k_1}, \ldots, g_{k_\ell}$ each will have a fan-out of FO ($|D_1| = |D_2| = \ldots = |D_\ell| = FO$). In this case, g_i will still have a fan-out problem (although less than the original problem) following the transformation, and further transformations will have to be employed.

<u>Transformation T4</u>: Consider the two gates shown in Fig. 3.2-6 in which gate g_i has a fan-in problem and there exists a gate g_j such that $IP(G_j) \subset IP(g_i)$. If condition C4 is satisfied, then a new gate g_k is added as shown in Fig. 3.2-7. Gate g_k's output is connected as an input to gate g_i, and the output of gate g_j is connected as an input to gate g_k. Finally, every connection from an immediate predecessor of g_j is removed from g_i.

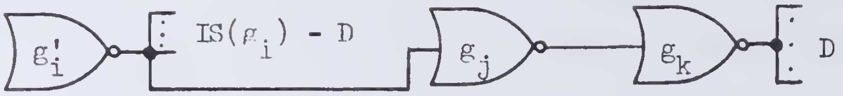


Fig. 3.2-3 Gate g_i after application of T3 when FO $< |IS(g_i)| \leq$ $(2 \times FO) - 1$. After the transformation: $|IS(g_i) - D + 1|$ = FO and $|D| \leq$ FO.

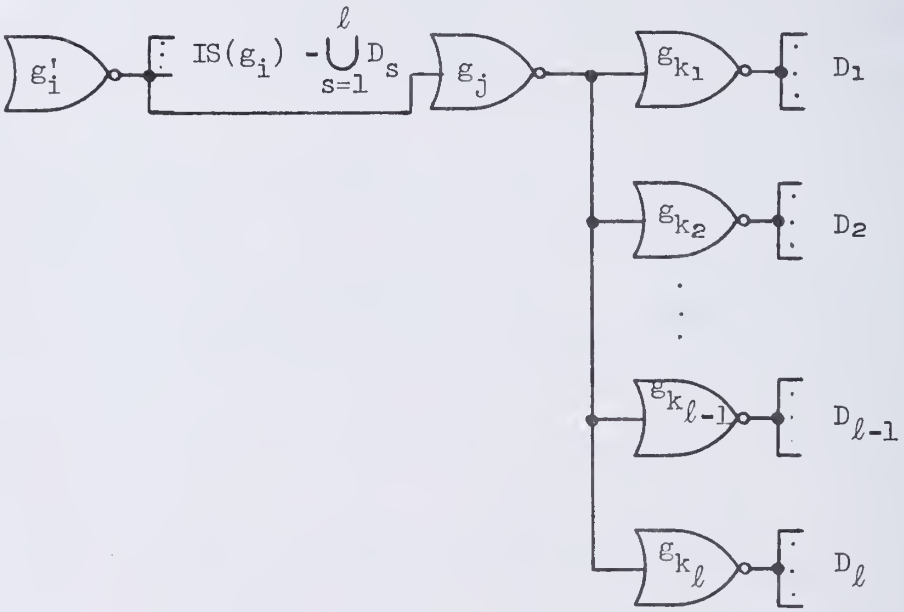


Fig. 3.2-4 Gate g_i after application of T3 when $(2 \times FO) - 1 <$ $|IS(g_i)| \leq (FO)^2 + FO - 1$. After the transformation: $|D_1| = |D_2| = \ldots = |D_{\ell-1}|$ = FO, $|D_\ell| = |IS(g_i)| - (\ell \times FO) + 1 \leq$ FO, $|IS(g_i')| = |IS(g_i) - \bigcup_{s=1}^{\ell} D_s| + 1 =$ FO, and $\ell \leq$ FO.

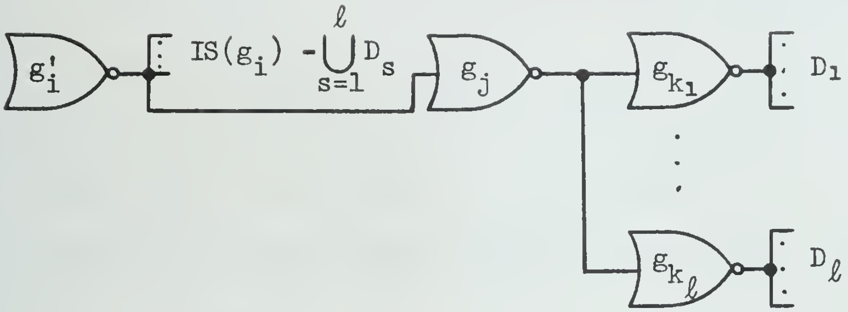


Fig. 3.2-5 Gate g_i after application of T3 when $|IS(g_i)| > (FO)^2 + FO - 1$. After the transformation: $|D_1| = |D_2| = \ldots = |D_\ell|$, $\ell = FO$ and $|IS(g_i) - \bigcup_{s=1}^{\ell} D_s| + 1 > FO$.

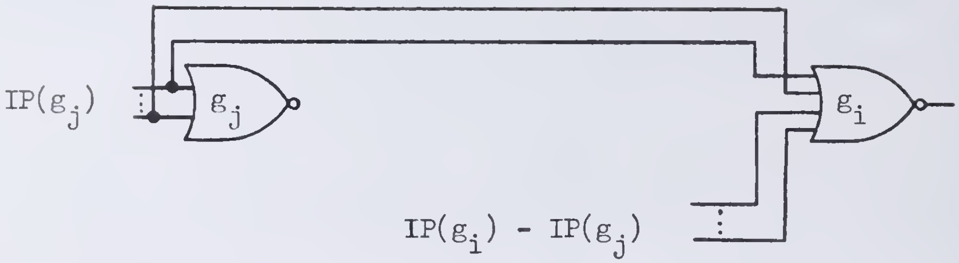


Fig. 3.2-6 Gate g_i has a fan-in problem and $IP(g_j) \subset IP(g_i)$ holds.

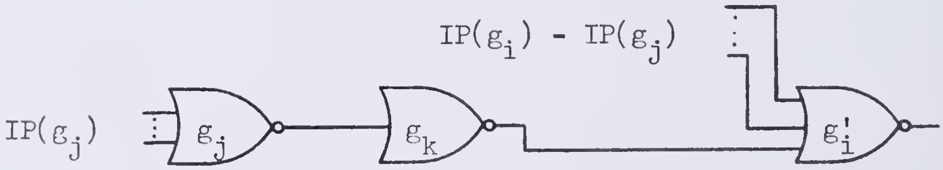


Fig. 3.2-7 The fan-in problem of g_i is solved by transformation T4. After the transformation: $|IP(g_i) - IP(g_j)| + 1 \leq FI$.

Transformation T5: This transformation is used to solve the fan-in problem of a gate g_i when C4 cannot be satisfied. There are three general cases:

(1) If gate g_i has a fan-in problem and $FI < |IP(g_i)| \leq (2 \times FI) - 1$, then the corresponding transformation results in Fig. 3.2-8. An output from a new gate, g_j, is first connected to g_i. As many input connections as necessary to correct g_i's fan-in problem are transferred from g_i to a new gate g_k and then the output of gate g_k is connected to g_j (in Fig. 3.2-8, D designates the set of gates whose output connections are transferred).

(2) If gate g_i has a fan-in problem and $(2 \times FI) - 1 < |IP(g_i)| \leq (FI)^2$, then the corresponding transformation results in Fig. 3.2-9. This transformation and that shown in Fig. 3.2-8 are similar, but in this case gate g_i is fed by ℓ new gates $g_{j_1}, g_{j_2}, \ldots, g_{j_\ell}$ where ℓ is the smallest integer such that $\ell \times (FI - 1) + FI \geq |IP(g_i)|$. For each gate g_{j_p} $(p = 1, 2, \ldots, \ell)$, the output of another new gate, g_{k_p} is connected to g_{j_p}. Input connections are transferred from g_i to $g_{k_1}, g_{k_2}, \ldots, g_{k_\ell}$ such that $g_i, g_{k_1}, g_{k_2}, \ldots, g_{k_{\ell - 1}}$ each will have a fan-in of FI while g_{k_ℓ} will have a fan-in of $|IP(g_i)| - \ell \times (FI - 1)$ (which will be $\leq$ FI). Here the set of connections transferred from g_i to g_{k_s} $(s = 1, \ldots, \ell)$ is designated D_s $(s = 1, \ldots, \ell)$, respectively.

(3) If gate g_i has a fan-in problem and $|IP(g_i)| > (FI)^2$, then the corresponding transformation results in Fig. 3.2-10. Outputs from ℓ new gates $g_{j_1}, g_{j_2}, \ldots, g_{j_\ell}$, where $\ell = FI$, are connected to g_i. Again, for each gate g_{j_p} $(p = 1, 2, \ldots, \ell)$, the output of another new gate, g_{k_p}, is connected to g_{j_p}. Input connections are transferred from g_i to $g_{k_1}, g_{k_2}, \ldots, g_{k_\ell}$ such that $g_{k_1}, g_{k_2}, \ldots, g_{k_\ell}$ each will have a fan-in of FI. The only difference between this transformation and that shown in Fig. 3.2-9 is that gate g_i will still be

left with a fan-in problem after the transformation (although less severe than the original problem) and further transformation will have to be employed.

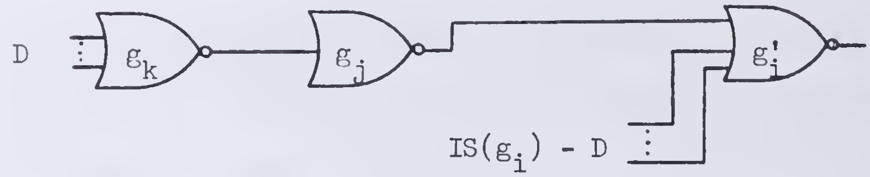


Fig. 3.2-8 Gate g_i after application of T5 when $FI < |IP(g_i)| \leq (2 \times FI) - 1$. After the transformation: $|IS(g_i) - D + 1| = FI$ and $|D| \leq FI$.

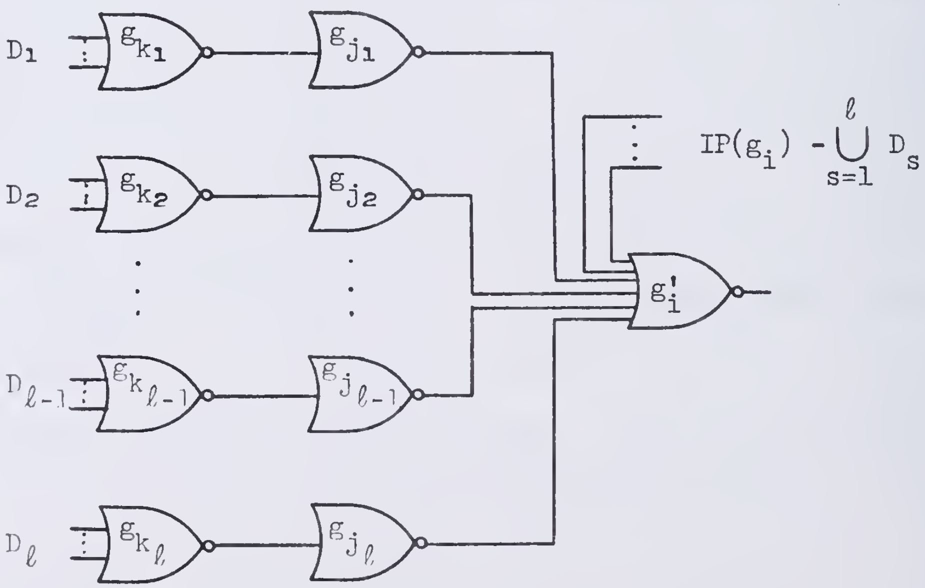


Fig. 3.2-9 Gate g_i after application of T5 when $(2 \times FI) - 1 < |IP(g_i)| \leq FI^2$. After the transformation: $|D_1| = |D_2| = \ldots = |D_{\ell-1}| = FI$, $|D_\ell| = |IP(g_i)| - \ell \times (FI - 1) \leq FI$, $|IP(g_i')| = |IP(g_i) - \bigcup_{s=1}^{\ell} D_s| + \ell = FI$, and $\ell \leq FI$.

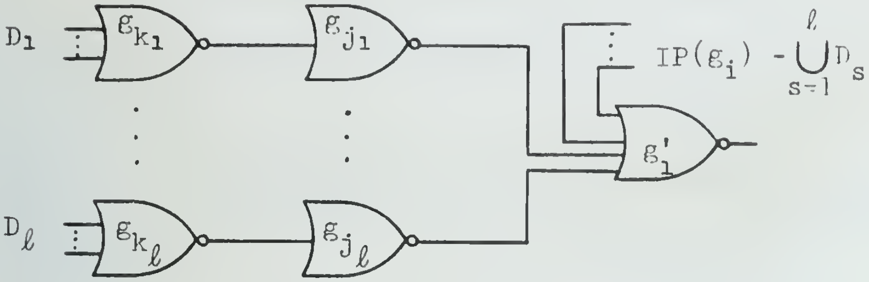


Fig. 3.2-10 Gate g_i after application of T5 when $|IP(g_i)| > (FI)^2$. After the transformation: $|D_1| = |D_2| = \ldots = |D_\ell| = FI$, $\ell = FI$ and $|IP(g_i')| = |IP(g_i) - \bigcup_{s=1}^{\ell} D_s| + \ell = |IP(g_i)| - (FI)^2 + FI > FI$.

Transformation T6: This transformation is applied when condition C6 is satisfied.

For example, consider the original subnetwork configuration in Fig. 3.2-11 in which there exists a set of gates $K = \{g_{i_1}, g_{i_2}, \ldots, g_{i_k}\}$ and a set of gates $H = \{g_{j_1}, g_{j_2}, \ldots, g_{j_h}\}$ such that the output of each gate g_i is connected to each gate g_j. There are two general cases:

(1) If the subnetwork in Fig. 3.2-11 exists and satisfies condition set C6 and $h \leq FO$, then the corresponding transformation results in Fig. 3.2-12. An output from $g_{i_1}, g_{i_2}, \ldots, g_{i_k}$ is connected to a new gate g_p. The output of g_p in turn is connected to another new gate g_q. Gate g_q then is connected as an input to $g_{j_1}, g_{j_2}, \ldots, g_{j_h}$. The h output connections from each of gates $g_{i_1}, g_{i_2}, \ldots, g_{i_k}$ to gates $g_{j_1}, g_{j_2}, \ldots, g_{j_k}$ are removed.

(2) If the subnetwork in Fig. 3.2-11 exists and satisfies condition set C6 and FO < h $\leq$ $(FO)^2$, then the corresponding transformation results in

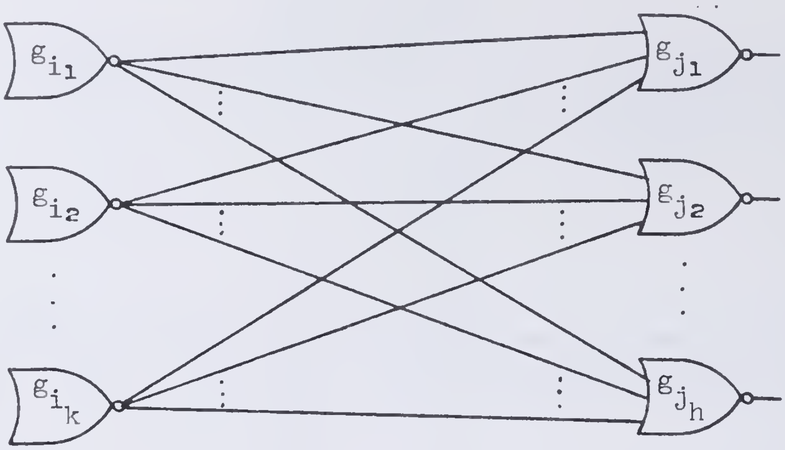
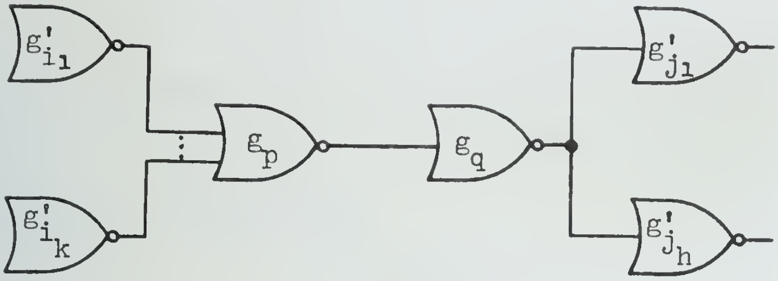


Fig. 3.2-11 Subnetwork structure upon which transformation T6 may be applied.

Fig. 3.2-13. The only difference between this transformation and that shown in Fig. 3.2-12 is that gate g_p now fans out to ℓ newly added gates g_{q_1}, g_{q_2}, ..., g_{q_ℓ} where ℓ is the smallest integer such that $\ell \times FO \geq h$. Output connections from each of g_{q_1}, g_{q_2}, ..., $g_{q_{\ell - 1}}$ will be fed to FO of the g_j's (i.e. g_{q_1} will feed g_{j_1}, g_{j_2}, ..., $g_{j_{FO}}$; g_{q_2} will feed $g_{j_{FO + 1}}$, $g_{j_{FO + 2}}$, ..., $g_{j_{2FO}}$; etc.). Gate g_{q_ℓ} will fan-out to the h - (ℓ - 1) FO gates: $g_{j_{(\ell - 1) FO + 1}}$, ..., g_{j_h}. The h output connections from each of gates g_{i_1}, g_{i_2}, ..., g_{i_k} to gates g_{j_1}, g_{j_2}, ..., g_{j_k} are removed.

After T6 is applied at least one of the gates in Fig. 3.2-11 has its fan-in or fan-out problem completely solved while the fan-in and fan-out problems of the rest of the gates may only be partially solved.

The transformation methods were implemented as a transformation program by J. G. Legge. Many problems were run to test the effectiveness of these transformations. Table 3.2-1 gives the average percentage of the number



ig. 3.2-12 Subnetwork structure (Fig. 3.2-11) after the application of T6 when $h \leq FO$. After the transformation: $|IS(g'_{i_1})| = |IS(g_{i_1})| + 1 - h$, $|IS(g'_{i_2})| = |IS(g_{i_2})| + 1 - h$, ... , $|IS(g'_{i_k})| = |IS(g_{i_k})| + 1 - h$. $|IP(g'_j)| = |IP(g_j)| + 1 - k$, $|IP(g'_{j_2})| = |IP(g_{j_2})| + 1 - k$, ... , $|IP(g'_{j_h})| = |IP(g_{j_h})| + 1 - k$.

of times a certain transformation was applicable during a complete run of test networks for 30 4-variable functions and 30 5-variable functions for a given set of fan-in/fan-out restrictions. (For given networks, some transformations cannot be applied because the corresponding conditions are not met.) This table shows that T5 was applicable in a greater percentage of cases (39%) than any other transformation (this means that on the average 39% of fan-in, fan-out problems occurred in a complete run are suitable for T5 to solve), and T1 and T4 were not applicable very often (this is because that the special conditions which must be satisfied were not often satisfied).

More details can be found in [24].

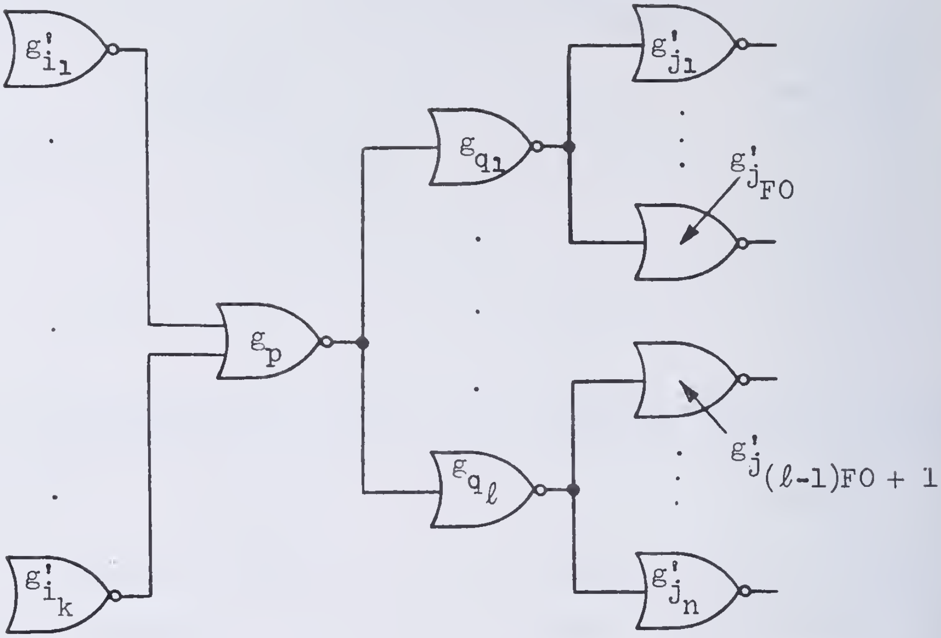


Fig. 3.2-13 Subnetwork structure (Fig. 2.3.1.1) after the application of T6 when FO < h $\leq$ $(FO)^2$. After the transformation: $|IS(g'_{i_1})| = |IS(g_{i_1})| + 1 - h$, $|IS(g'_{i_2})| = |IS(g_{i_2})| + 1 - h$, ... , $|IS(g'_{i_k})| = |IS(g_{i_k})| + 1 - h$. $|IP(g'_{j_1})| = |IP(g_{j_1})| + 1 - k$, $|IP(g'_{j_2})| = |IP(g_{j_2})| + 1 - k$, ... , $|IP(g'_{j_h})| = |IP(g_{j_h})| + 1 - k$.

Table 3.2-1 The percentage of the number of times a transformation was applied during a complete test run.

Transformation	T1	T2	T3	T4	T5	T6
percentage	0.7	12.1	19.6	3.0	39.0	25.6

3.3 Transduction Procedures

In this section we provide a summary of all transduction procedures. The details about the transduction procedures can be found in the references given in the following subsections.

3.3.1 Basic definitions and ideas

Let $X = \{x_1,x_2,\ldots,x_n\}$, $Z = \{z_1,z_2,\ldots,z_m\}$, $V_I = \{v_1,v_2,\ldots,v_p\}$, $V_G = \{v_{p+1},v_{p+2},\ldots,v_{p+R}\}$ and $V_0 = \{v_{p+R+1},v_{p+R+2},\ldots,v_{p+R+m}\}$ be the set of n external variables, the set of m output functions, the set of p input terminals, the set of R NOR gates, and the set of m output terminals, respectively. For simplicity we can assume only uncomplemented external variables (x_i's) are available as input variagles, i.e., $p = n$. Let $C=\{c_{ij}\}$ be the set of connections, where c_{ij} denotes a connection fed from $v_i \varepsilon V_I \vee V_G \vee V_0$ to $v_j \varepsilon V_0 \vee V_G$. Let $V = V_I \vee V_G \vee V_0$. If there exists a sequence of gates v_{k1}, v_{k2}, ..., v_{kt} between v_i and v_j with a non-negative integer t such that $v_{k1} \varepsilon IS(v_i)$, $v_{kq} \varepsilon IS^{*}(v_{k,q-1})$ for q = 2, 3, ..., t and $v_j \varepsilon IS(v_{kt})$, then v_i is a predecessor of v_j and v_j is a successor of v_i. Let $P(v_i)$ and $S(v_i)$ denote the set of all predecessors of v_i and the set of all successors of v_i, respectively.

A function f_i (completely specified or incompletely specified),

*The definition of immediate successors and immediate predecessors are given in the previous section.

realized at an input terminal, the output of a gate, or a connection (more precisely speaking, the function realized at the input of a next gate to which the connection feeds), is represented by a 2^n dimensional vector

$$f_i = (f_i^{(1)}, f_i^{(2)}, \ldots, f_i^{(2^n)})$$

where
$$f_i^{(j)} = 1 \text{ if } f_i(x_1, x_2, \ldots, x_n) = 1,$$
$$f_i^{(j)} = 0 \text{ if } f_i(x_1, x_2, \ldots, x_n) = 0,$$
$$f_i^{(j)} = * \text{ if } f_i(x_1, x_2, \ldots, x_n) = d \begin{pmatrix}\text{don't}\\ \text{care}\end{pmatrix}$$

for $j - 1 = 2^{n-1} x_1 + 2^{n-2} x_2 + \ldots + x_n.$

External variables are represented as follows:

$$x_1 = (\underbrace{0, 0, \ldots \ldots, 0,}_{2^{n-1}} \underbrace{1, 1, \ldots \ldots, 1}_{2^{n-1}}),$$

$$x_2 = (\underbrace{0, \ldots, 0,}_{2^{n-2}} \underbrace{1, \ldots, 1,}_{2^{n-2}} \underbrace{0, \ldots, 0,}_{2^{n-2}} \underbrace{1, \ldots, 1}_{2^{n-2}}),$$

$$\vdots$$

$$x_{n-1} = (0, 0, 1, 1, \ldots, \ldots, 0, 0, 1, 1),$$
$$x_n = (0, 1, 0, 1, \ldots, \ldots, 0, 1, 0, 1).$$

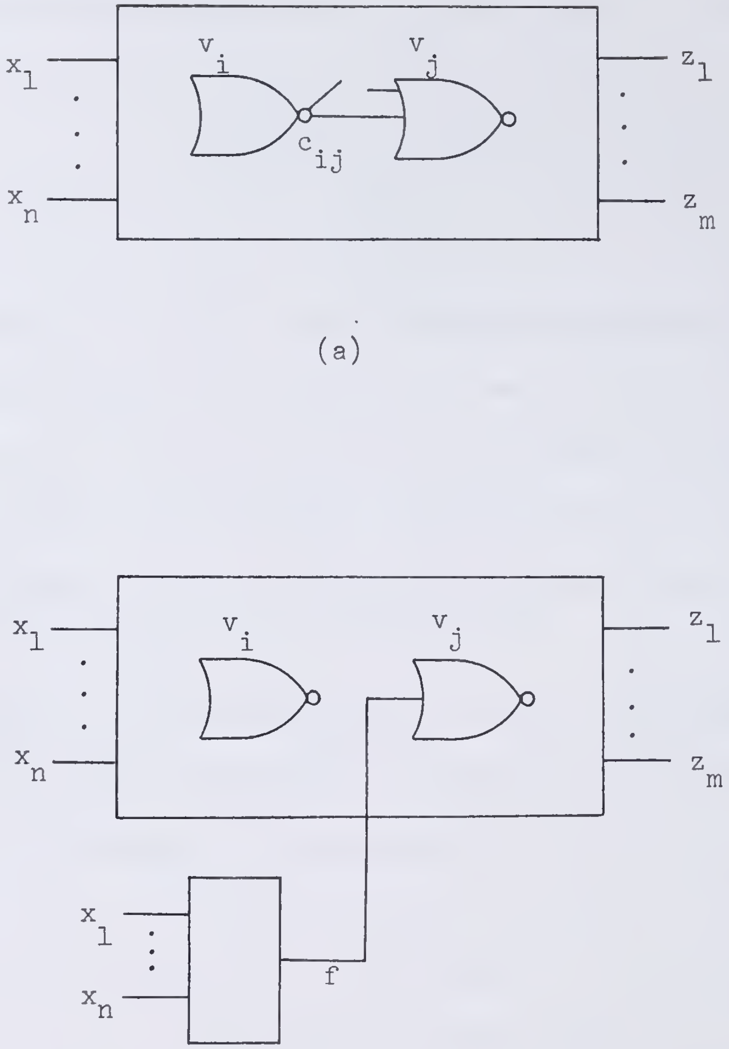
Definition 3.3.1-1 - If no specified components of the network output functions change by replacing the function realized at input terminal v_i, gate v_j or connection c_{ij}, by a function f, then function f is called a permissible function for input terminal v_i, gate v_j or connection c_{ij}, respectively.

This is illustrated in Fig. 3.3.1-1. Suppose we want to find a permissible function for the function realized at the right end of connection c_{ij} in the network in (a). When we replace the function realized at c_{ij} by a function f of variables x_1, x_2, ..., x_n as shown in (b), f is a permissible function for c_{ij} if specified components of z_1, ..., z_m in (b) are not different from those in (a). Usually there is more than one permissible function for c_{ij}.

Notice that permissible functions for a connection c_{ij} are separately defined from those for a gate v_i with the same i. Therefore, when gate v_i has more than one fan-out connection, there may be generally some permissible functions of v_i which are not permissible functions of c_{ij}; and there may be generally some permissible functions of c_{ij} which are not permissible functions of v_i.

Definition 3.3.1-2 - The set of all permissible functions for any input terminal, gate or connection can be consequently expressed by a single vector [17]. This set is called the maximum set of permissible functions, abbreviated as MSPF. Let $G_M(v_i)$ and $G_M(c_{ij})$ denote maximum sets of permissible functions for v_i and c_{ij}, respectively. Let $G(v_i)$ and $G(c_{ij})$ denote arbitrary subsets of $G_M(v_i)$ and $G_M(c_{ij})$, respectively.

Definition 3.3.1-3 - A gate or a connection is said to be S-redundant (implying single-redundant) if no specified components of the network outputs change by removing the gate or the connection. In some networks, no specified components of the network outputs change if two or more gates and/or connections are removed while the outputs change if a single or a connection is removed. In contrast to "S-redundant," it is called M-redundant (implying multiple-redundant). A network without any S-redundant gates or connections is called S-irredundant.



(a)

(b)

Fig. 3.3.1-1 Permissible function gor connection c_{ij}.

There is a procedure (Procedure MSPF) which can calculate MSPF for a given network, and there is another procedure (Procedure SINM) which can obtain S-irredundant networks based on the MSPF's [17]. But the calculation of the MSPF's is time-consuming and in Procedure SINM each time an S-redundant connection is removed, the recalculation of MSPFs for the entire new network is required. Thus the concept of a compatible set of permissible functions is introduced to develop more practical procedures.

Definition 3.3.1-4 - A tied subnetwork of a given network is defined as part of the given network satisfying the following conditions, where U is the set of input terminals, gates, connections and output terminals, contained in this subnetwork.

(1) If $v_i \varepsilon U$, then $S(v_i) \subseteq U$,

(2) If $c_{ij} \varepsilon U$, then $v_j \varepsilon U$ and $S(v_j) \subseteq U$,

(3) If $v_i \varepsilon U$, $v_j \varepsilon U$ and $v_j \varepsilon IS(v_i)$, then $c_{ij} \varepsilon U$.

Definition 3.3.1-5 - Let U be the set in input terminals, gates, connections and output terminals in a tied subnetwork N of a given network. All the sets of permissible functions, $G(v_i)$ and $G(c_{ij})$ ($v_i \varepsilon U$, $c_{ij} \varepsilon U$), are said to be compatible with respect to this tied subnetwork if the following property holds with every subset W of U.

(1) Replace the function at each element w in W by a function in G(w).

(2) In the resultant new network each element u in U, which is not contained in W, realizes some function contained in G(u).

(3) The above condition (2) holds, no matter which function G(w) is chosen for each w in condition (1) (i.e., condition (2) holds for every different choice of functions in G(w)'s for all the w's of W).

As a special case, a tied subnetwork can be a given network itself. In this case if sets of permissible functions satisfy the above conditions, they are simply called compatible sets of permissible functions (CSPF's). CSPF for an element u (ε V U C) is denoted by $G_C(u)$.

3.3.2 Pruning Procedures

For any given network, by comparing the outputs of the original network with the outputs of the network without a particular connection we can determine whether or not this connection is reduncant. This idea is simple but a computer program implementing this idea is too time-consuming to execute. So in the pruning procedures, we calculate the permissible functions of a selected gate or the input connections of a selected gate to decide which input connections of the selected gate can be removed. Three different ways are used to prune redundant connections:

(I) Pruning procedure based on CSPF's

For any given network this procedure calculates a set of CSPF's. Since the set of CSPF's is not unique for a non-trivial given network, it is desirable to choose a set of CSPF's such that as many CSPF's as possible consist of only 0s and *s. Those connections c_{ij} with $G_c(c_{ij})$ consisting of only 0s and *s are redundant and hence can be removed. The detailed steps for pruning redundant connections are given in [2].

It is observed that the pruning procedure based on CSPF's takes very short computation time. Since only a sufficient condition for a redundancy of a connection is used, a redundant connection sometimes cannot be detected by this procedure.

(II) Pruning procedure based on 0-fixed maximum set of permissible functions

A 0-fixed maximum set of permissible functions (OFMSPF) $G_{OM}(c_{ij})$ for a connection c_{ij} is defined as a subset of MSPF $GM(c_{ij})$ satisfying the following condition:

$$\text{for } f^{(d)}(v_i) = 0 \text{ , } G^{(d)}_{OM}(c_{ij}) = 0 \text{ holds,}$$
$$\text{and for } f^{(d)}(v_i) = 1 \text{ , } G^{(d)}_{OM}(c_{ij}) = G^{(d)}_{M}(c_{ij}) \text{ holds.}$$

In other words, the components of the OFMSF of a connection c_{ij} are found first for the components of $f(v_i)$ which are fixed to 0 and then found for other components of $f(v_i)$ by calculating the MSPF for c_{ij}.

It is proved that a connection c_{ij} is S-redundant if and only if the OFMSPF $G_{OM}(c_{ij})$ consists of 0s and *s [2]. The pruning procedure based on OFMSPF selects one connection c_{ij} at a time according to a particular ordering and checks whether $G_{OM}(c_{ij})$ contains only 0s and *s. If this is true, then remove c_{ij} and go to find another c_{ij}; otherwise go to find another c_{ij}. These steps will be applied until no further pruning is possible.

(III) Pruning procedure based on 1-fixed maximum set of permissible functions

A 1-fixed maximum set of permissible functions (1FMSPF) $G_{1M}(v_i)$ for for gate v_i is defined to be a subset of $G_M(v_i)$ satisfying the following condition:

$$\text{for } f^{(d)}(v_i) = 0 \text{ , } G^{(d)}_{1M}(v_i) = G^{(d)}_{M}(v_i) \text{ holds,}$$
$$\text{and for } f^{(d)}(v_i) = 1 \text{ , } G^{(d)}_{1M}(v_i) = 1 \text{ holds.}$$

In other words, the components of the 1FMSPF of a gate v_i are found first for the components of $f(v_i)$ which are fixed to 1 and then found for other components of $f(v_i)$ by calculating the MSPF for v_i.

It is proved that a connection c_{ij} is S-redundant if and only if for every d such that $G^{(d)}_{1M}(v_j) = 0$, $\bigvee_{\substack{v_k \varepsilon IP(v_j) \\ v_k \neq v_i}} f^{(d)}(v_k) = 1$ holds [2]. The pruning procedure based on 1FMSPF selects one gate $v_j \varepsilon V_G$ at a time according to a particular ordering and checks whether for every d such that $G^{(d)}_{1M}(v_j) = 0$, $\bigvee_{\substack{v_k \varepsilon IP(v_j) \\ v_k \neq v_i}} f^{(d)}(v_k) = 1$. If this is true, then remove c_{ij} and go back to find another v_j; otherwise go back to find another v_j. These steps will be repeated until no further pruning is possible.

The pruning procedures aim at removing redundant connections only; but sometimes because of the removal of redundant connections, some gates also become redundant and hence can be removed.

The pruning procedures (I), (II) and (III) were implemented in the transduction programs NETTRA-PG1*, NETTRA-P1 and NETTRA-P2, respectively. Usually, procedures (II) and (III) take longer computation time than procedure (I), but after applying either procedure (II) or procedure (III), an S-irredundant network is always obtained.

3.3.3 Procedures based on gate merging

Two gates v_i and v_j are said to be mergeable if the following condition are satisfied:

*NETTRA-PG1 also realizes the simple substituting procedure, and this will be explained later.

(1) $v_j \notin S(v_i)$ and $v_i \notin S(v_j)$ (This guarantees that the resultant network is loop-free.)

(2) Each output terminal v_{n+R+i} realizes a different function in set z_i for $i = 1,2, \ldots, m$, after the following network reconfigurations:

(2-1) Add connections from all gates and input terminals in $IP(v_j) - IP(v_i)$ to gate v_i.

(2-2) Add connections from all gates and input terminals in $IP(v_i) - IP(v_j)$ to gate v_j.

If two gates v_i and v_j are mergeable, then we can replace these two gates by gate v_{ij} satisfying

$$IP(v_{ij}) = IP(v_i) \vee IP(v_j),$$

$$IS(v_{ij}) = IS(v_i) \vee IS(v_j).$$

Gate v_{ij} is called the merged gate.

A simple example is shown below:

Example 3.3.3-1 - The network given in Fig. 3.3.3-1(a) realizes function $f = x_1 x_2 \vee \bar{x}_1 \bar{x}_2$. We can add redundant inputs x_2 and x_1 to gates v_6 and v_7, respectively without changing the output of v_3. By adding these redundant inputs we find that v_6 and v_7 are mergeable. The network obtained by merging gates v_6 and v_7 is shown in Fig. 3.3.3-1(b). Gate $v_{6,7}$ is the merged gate.

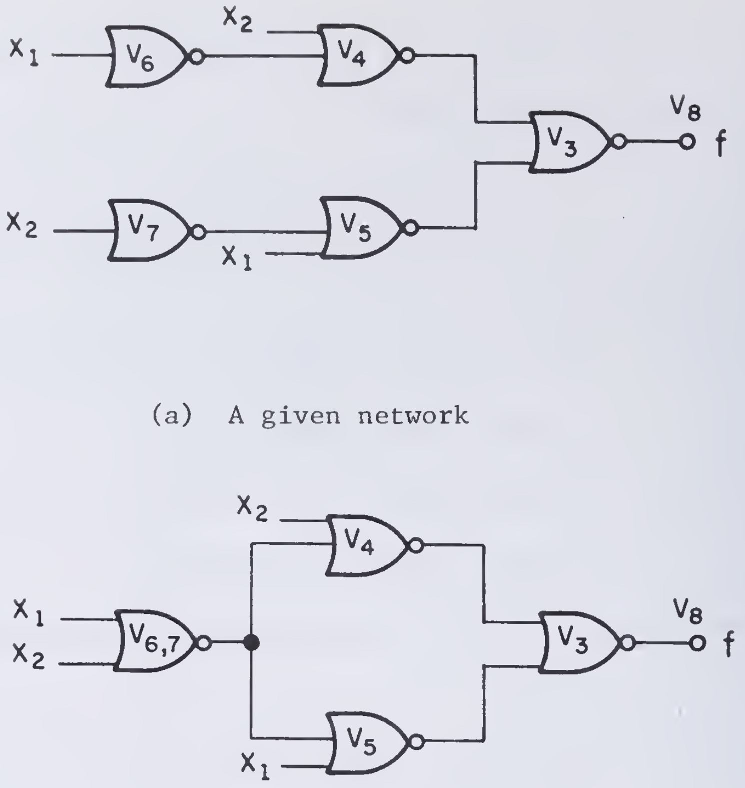
It is proved that gate v_i and v_j are mergeable if the following conditions are satisfied [22]:

(1) $v_j \notin S(v_i)$ and $v_i \notin S(v_i)$

(2) $G_c(v_i) \cap G_c(v_j) \neq \phi$

(3) $f(v_i) \cdot f(v_j) \varepsilon G_c(v_i) \cap G_c(v_j)$,

where symbol $\cdot$ denotes the logical AND operation for each component of $f(v_i)$ and $f(v_j)$.



(a) A given network

(b) Simplified network

Fig. 3.3.3-1 A simple example for gate merging

The following definitions and theorems are needed for further descriptions:

Definition 3.3.3-1 - A gate or an input terminal, v_i, is said to be connectable to a gate v_j with respect to a set of permissible functions of v_j, $G(v_j)$, if (1) the output function of v_j remains in $G(v_j)$ after adding connection c_{ij}, and (2) the network obtained after this input addition is loop-free.

Definition 3.3.3-2 - Connection c_{ij} is said to be disconnectable from a gate v_j with respect to a set of permissible functions of v_j, $G(v_j)$ if the output function of v_j remains in $G(v_j)$ after removing c_{ij} from the network.

Definition 3.3.3-3 - The set of connectable functions to gate v_j with respect to $G(v_j)$, $K(v_j)$, is the set of all functions such that the addition of any subset of these functions to v_j as its inputs keeps the function realized at v_j remain in $G(v_j)$.

Theorem 3.3.3-1 - A gate or an input terminal, v_i, is connectable to v_j with respect to set $G(v_j)$ if and only if the following conditions are satisfied:

(1) For all d such that $f^{(d)}(v_i) = 1$,

$$G_c^{(d)}(v_j) = 0 \text{ or } *.$$

where $f^{(d)}(v_i)$ is the d-th component of the output vector of gate v_i.

(2) v_i is not contained in $S(v_j)$ where $S(v_j)$ denotes the set of successor gates of gate v_j.

Theorem 3.3.3-2 - Connection C_{ij} is disconnectable from gate v_j with respect to set $G(v_j)$ if and only if the following condition is satisfied: For all d such that $f^{(d)}(v_i) = 1$, either

$$G^{(d)}(v_j) = {}^* \text{ or}$$

$$\bigvee_{\substack{v \in IP(v_j) \\ v \neq v_i}} f^{(d)}(v) = 1$$

where $\vee$ denotes logical OR operation and $IP(v_j)$ denotes the set of all immediate predessor gates of gate v_j.

Theorem 3.3.3-3 - The set $K(v_j)$ of all connectable functions to a gate v_j with respect to $G(v_j)$ is given by

$K^{(d)}(v_j) = 0$ for all d such that $G^{(d)}(v_j) = 1$, and

$K^{(d)}(v_j) = {}^*$ for all other d's.

Now we are ready to review the procedures based on gate merging.

Procedure GMGC.

Step 1 Find two gates v_i and v_j such that

$$G_c(v_i) \cap G_c(v_j) \equiv G_c(v_{ij}) \neq \emptyset.$$

Step 2 Consider an imaginary gate v_{ij} whose CSPF is $G_c(v_{ij})$. Calculate the set $K(v_{ij})$ of all connectable functions to v_{ij}.

Step 3 Select the set U of gates and input terminals, v, satisfying the following conditions:

(a) For all $v \in U$, $f(v) \in K(v_{ij})$.

(b) For all $v \in U$, $v \notin S(v_i) \cup S(v_j)$

(loop-free condition).

Step 4 Connect all gates and input terminals in U to gate v_{ij}. If the output function is contained in $G_c(v_{ij})$, then go to Step 5; otherwise v_i and v_j cannot be replaced by gate v_{ij}.

Step 5 As

$$\overline{\bigvee_{v \in U} f(v)} \in G_c(v_{ij})$$

holds, using the disconnectable condition select a new input set U' (which is a subset of U) such that

$$\overline{\bigvee_{v \in U'} f(v)} \in G_c(v_{ij}).$$

Step 6 Replace v_i and v_j by gate v_{ij} which has input connections from all the gates and input terminals in U'.

More general procedures are discussed in detail in [22]. The transduction procedures based on gate merging are implemented in the transduction program NETTRA-G3.

3.3.4 Procedures based on gate substitution

For any given network if the disjunction of the outputs of some gates and external variables is found to be identical to the output of a gate v_i, then we can substitute the disjunction of these outputs of gates and/or external variables for the output connections of gate v_i. A simple example is shown in Fig. 3.3.4-1. Assume the disjunction of outputs of v_{j1}, v_{j2} and external variable x_ℓ is identical to the output of v_i, i.e.,

$$f(v_{j1}) \vee f(v_{j2}) \vee x_\ell = f(v_i).$$

Then, the connections from v_i to v_{k1} and v_{k2} can be replaced by the connections from v_{j1}, v_{j2} and x_ℓ to v_{k1} and v_{k2} as shown in Fig. 3.3.4-1(b).

The following is a procedure for the substitution of a gate.

Procedure SGC. Substitution of a gate using CSPF's.

Step 1 Calculate CSPF's for all gates in a given network.

Step 2 Select one gate v_i.

(2-1) Calculate a set $H(v_i)$ which is defined by

$$H(v_i) = \bigcap_{v \varepsilon IS(v_i)} K(v),$$

where $K(v)$ is the set of connectable functions to gate v with respect to the CSPF for v (i.e., $G_c(v)$).

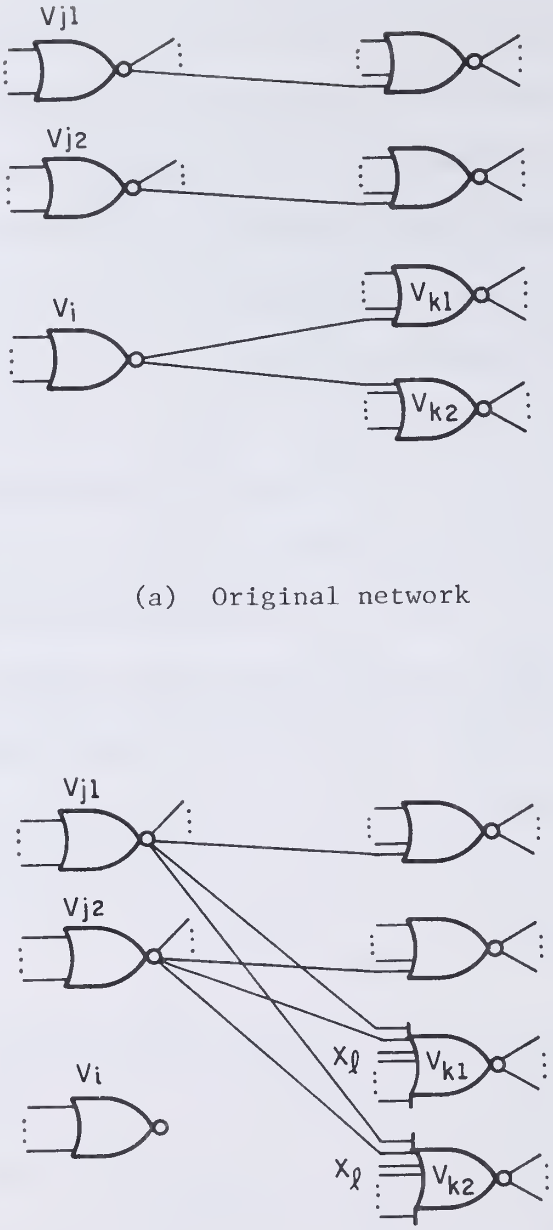
(2-2) Let U be a set of all gates and input terminals satisfying

$$v \notin S(v_i), \text{ and}$$

$$f(v) \varepsilon H(v_i) \text{ for every } v \varepsilon U.$$

(2-3) If $\bigvee_{v \varepsilon U} f(v) \varepsilon G_c(v_i)$, then go to Step 3; otherwise repeat Step 2 until all gates are considered.

Step 3 Substitute connections from gates and input terminals in U for the output connections of gate v_i.



(a) Original network

(b) After gate substitution

Fig. 3.3.4-1 Simple example for gate substitution

Instead of substituting outputs of gates and input terminals for gate v_i, we can generalize the procedure by substituting outputs of gates and input terminals for each output of gate v_i (possibly a different set of outputs of gates and input terminals for each output of v_i). A simple example is shown in Fig. 3.3.4-2. Gate v_{12} of the network shown in Fig. 3.3.4-2(a) has three outputs connecting to v_6, v_7 and v_8. These connections, however, can be replaced by connections from v_{11}, v_9 and v_{10}, respectively. So we can remove gate v_{12} (see Figure 3.2(b)). The following procedure is a generalization of Procedure SGC.

Procedure SOGC. Substitution of outputs of a gate using CSPF's.

Step 1 Select one gate v_i.

Step 2 Calculate CSPF's for all gates, input terminals and output connections of v_i according to some proper order.

Step 3 For each output connection c_{ij} of v_i, calculate a set U of gates and input terminals.

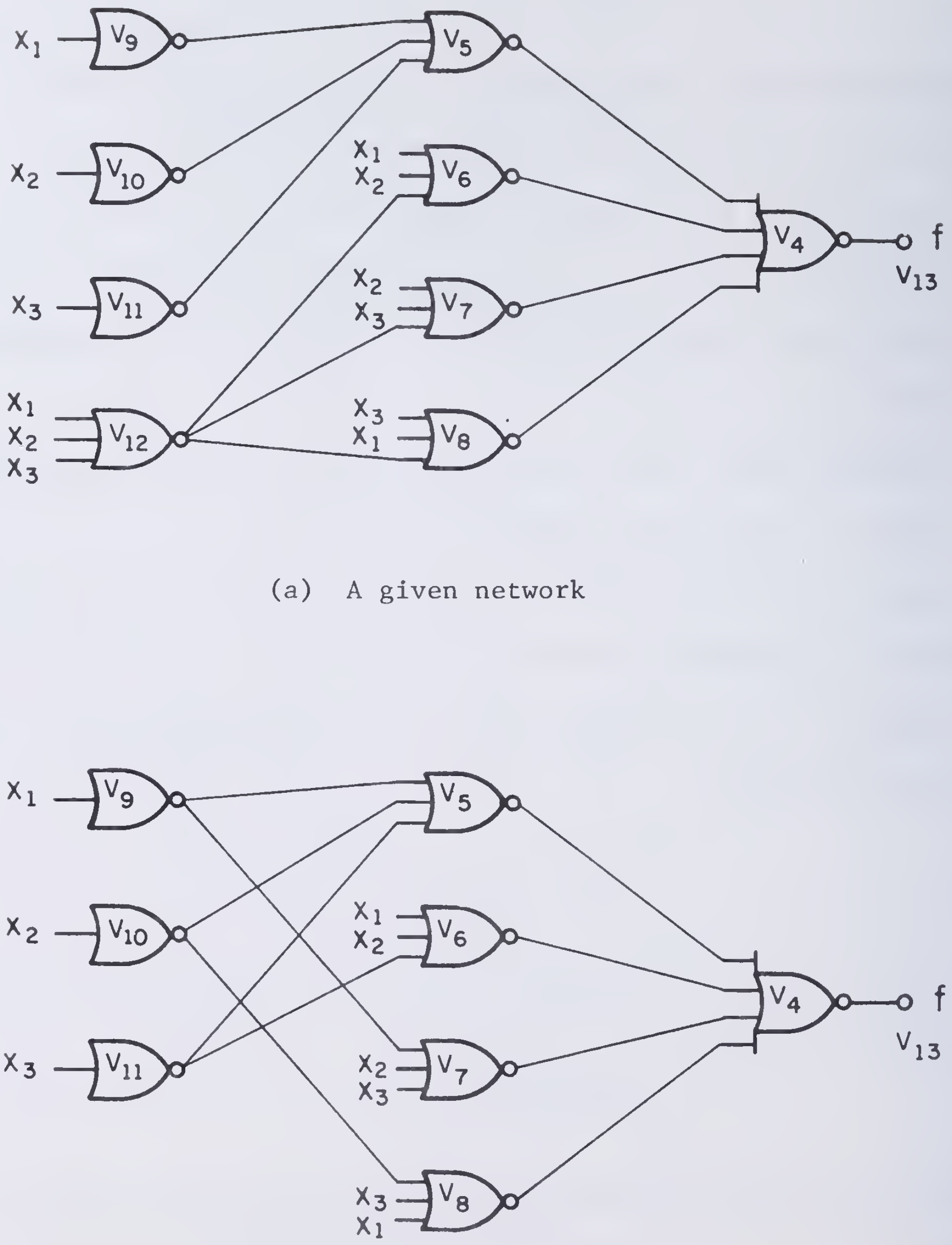
$$U = \{v | v \in V_G \cup V_I,\ v \notin S(v_i),$$
$$f(v) \in K(v_j)\}.$$

Step 4 If the following condition is satisfied then substitute connections from all elements in U to v_j for c_{ij}:

$$\bigvee_{v \in U} f(v) \in G_c(c_{ij}).$$

Disconnect some of these new connections by applying disconnectable conditions to gate v_j.

Step 5 Repeat Step 3 and Step 4 for all output connections of gate v_i. If there exists an output connection which cannot be substituted, then v_i cannot be removed. If so, repeat the procedure on the original network by selecting another gate.



(a) A given network

(b) After the substitution for each of the output connections of v_{12}

Fig. 3.3.4-2 A simple example for generalized gate substitution

The transduction procedures based on the substitution are implemented in the transduction program NETTRA-G4.

3.3.5 Procedures based on connectable and disconnectable functions

The definitions of connectable and disconnectable functions are introduced in section 3.3.3. The following is proved in [14]:

I. A gate or an input terminal, v_i, is connectable to a gate, v_j, with respect to set $G(v_j)$ if and only if the following conditions are satisfied:

(1) For all d such that $f^{(d)}(v_i) = 1$,

$$G^{(d)}(v_j) = 0 \text{ or } * .$$

where $f^{(d)}(v_i)$ is the d-th component of the output vector of gate v_i.

(2) v_i is not contained in $S(v_j)$ where $S(v_j)$ denotes the set of successor gates of gate v_j.

II. A connection c_{ij} is disconnectable from gate v_j with respect to set $G(v_j)$ if and only if the following condition is satisfied: For all d such that $f^{(d)}(v_i) = 1$,

either $$G^{(d)}(v_j) = * \text{ or}$$

$$\bigvee_{\substack{v \varepsilon IP(v_j) \\ v \neq v_i}} f^d(v) = 1$$

The transduction procedures based on connectable and disconnectable functions intend to replace the input functions of gates and to remove disconnectable inputs from gates. Some gates may become redundant after the removal of disconnectable inputs. An example is shown in Fig. 3.3.5-1, where the network realizes function $f = \bar{x}_2\bar{x}_3\bar{x}_4 \vee x_1\bar{x}_2\bar{x}_4 \vee x_1x_2\bar{x}_3 \vee \bar{x}_1x_2x_4 \vee \bar{x}_1x_3x_4 \vee \bar{x}_1x_2x_3$. After calculating the CSPFs of the network shown in Fig. 3.3.5-1(a), it is found that x_4 is connectable to gate 3 and gate 4 is connectable gate 5. After adding these connections, the connection between gate 8 and gate 5 is found to be disconnectable. The simplified network is shown in Fig. 3.3.5-1(b).

A procedure based on connectable and disconnectable functions is outlined below. The detailed descriptions are presented in [14].

Procedure based on connectable and disconnectable functions

Step 1: Select gate v_j according to some order. If all gates have been considered, then stop.

Step 2: Calculate the CSPF for v_j.

Step 3: Calculate a set $K(v_j)$ of connectable functions with respect to $G_c(v_j)$.

Step 4: Select external variables contained in $K(v_j)$ and select gates (not in $S(v_j)$) which realize functions in $K(v_j)$. Let Q be a set of these external variables and gates.

Step 5: Select a subset of Q_0 of Q such that every element in set Q_0 is essential for replacing the input connections of v_j. An external variable or a gate v_i in Q_0 is an essential input of v_j w.r.t $G_c(v_j)$ if and only if there exists at least one d such that:

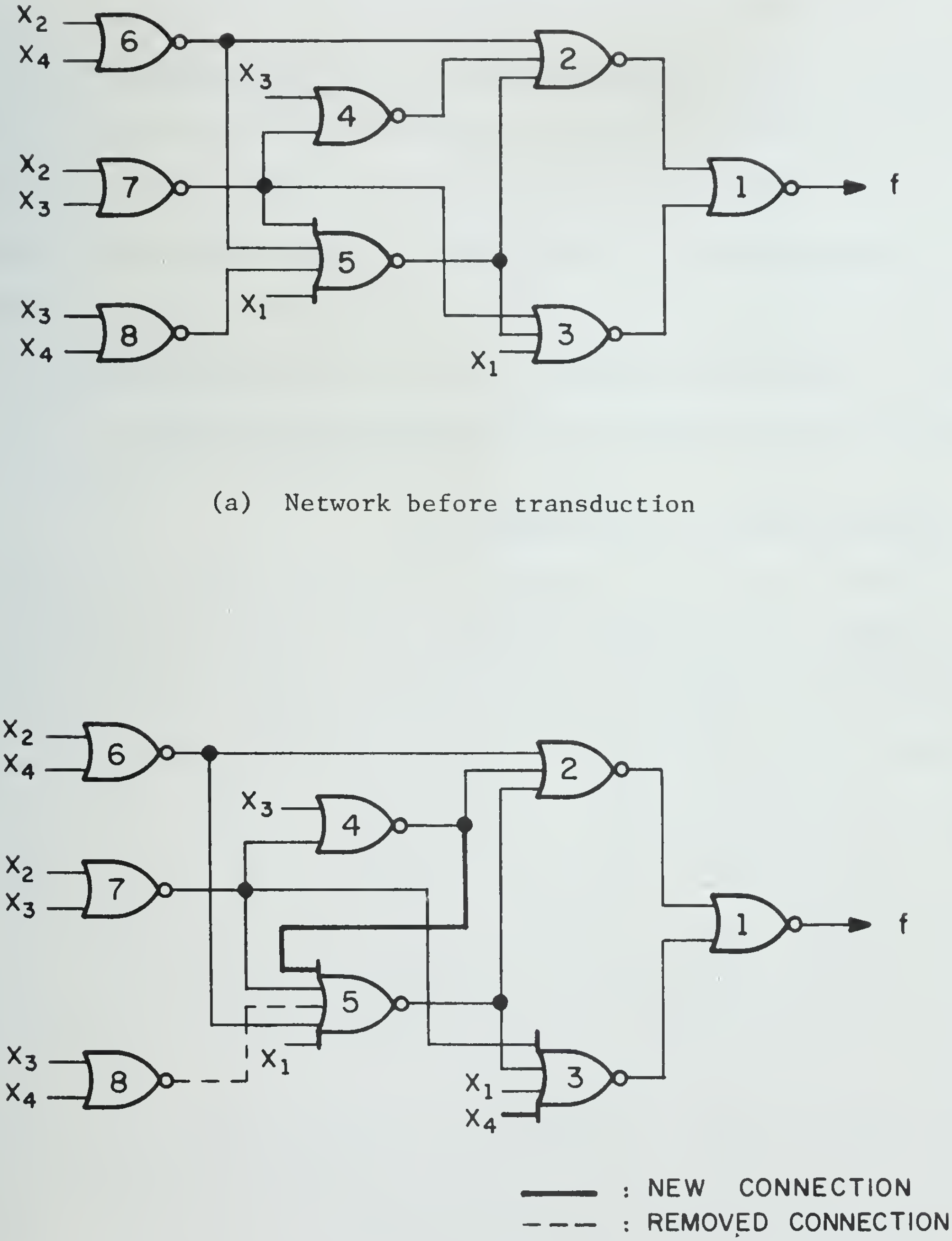
$$G_c^{(d)}(v_j) = 0, \quad f^{(d)}(v_i) = 1, \text{ and } \bigvee_{\substack{v \in Q_0(v_j) \\ v \neq v_i}} f^{(d)}(v) = 0$$

Step 6: Select a subset of Q' of Q such that

$$Q' \subseteq Q_0 \subseteq Q \text{ and}$$

$$\bigwedge_{v \in Q'} \overline{f}(v) \in G_c(v_j)$$

Step 7: Remove all connections which are inputs of v_j and connect all elements in Q' to gate v_j as inputs.



(a) Network before transduction

(b) Network after transduction

3.5-1 An example for procedures based on connectable and disconnectable functions

Step 8: Calculate the CSPF's for the modified network and remove all disconnectable connections. Remove gates which become redundant after the disconnectable connections are removed.

Step 9: If the current network has a lower cost than the original network, then go to step 1. Otherwise, select another Q' and go to step 6.

The transduction procedures based on connectable and disconnectable functions are implemented in the transduction programs NETTRA-G1 and NETTRA-G2. The primary difference between NETTRA-G1 and NETTRA-G2 is that NETTRA-G2 concentrates on removing specific gates from the network under consideration while NETTRA-G1 does not attempt to remove specific gates [1].

3.3.6 Procedures based on error-compensation

The basic idea in the transduction procedures based on error-compensation is simple. For any given network, a gate is selected according to an appropriate order. Assuming that this gate is removed, check the outputs of the network. If the outputs do not change, then the selected gate is redundant and can be actually removed. If the outputs do change, then try to compensate for these changes (errors) by reconfiguring the network. The selected gate becomes redundant if these changes can be compensated.

The following definitions are introduced to facilitate the later descriptions.

Definition 3.3.6-1 - A component of $G_c(v_i)$ for a gate v_i is called a 0-error (or $\underline{0}$) if it is originally a 1 and it changes to 0 because of the removal of other gates or connections in the network.

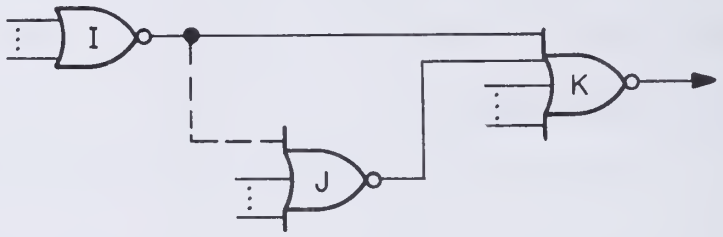
Definition 3.3.6-2 - A component of $G_c(v_i)$ for a gate v_i is called a 1-error (or $\underline{1}$) if it is originally a 0 and it changes to 1 because of the removal of other gates or connections in the network.

Definition 3.3.6-3 - A 0-component (or 0-error-component) of the output of gate v_i is covered by the input connection c_{ji} if the corresponding component of c_{ji} is a 1.

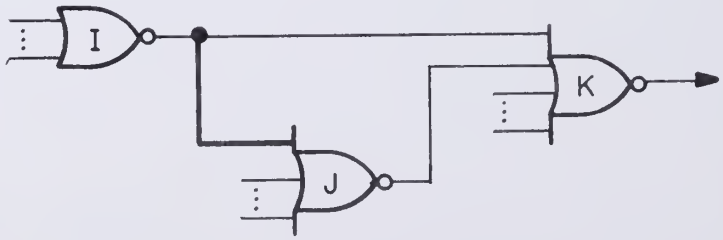
Definition 3.3.6-4 - The compatible set of permissible functions (CSPF) of a gate, an external variable or a connection is called a compatible set of permissible functions with errors (denoted by CSPFE) if it has some 1-error or 0-error components due to the removal of other gates and/or connections.

Definition 3.3.6-5 - A 0-error-component in the CSPFE of a gate v_i is called a primary 0-error-component if it is covered by only one input connection of this gate. A primary 0-error component is considered easier to be compensated.

In order to compensate for error-components in the CSPFE of a gate, functions currently realized at other gates and external variables may be used. But in order to make the error-compensation more flexible, the concepts of potential outputs and potential output table (POT) were introduced in [15], and a procedure utilizing a potential output table was also given. A potential output from a gate I is a function realized at I by connecting additional inputs to I. This potential output of I usually differs from the function currently realized at I, and therefore can be used to compensate for errors in a certain gate to which the current output function at I is not connectable. The principle used in generating potential output is called the triangular condition. In a loop-free NOR(NAND) network, suppose three gates are connected to each other to form a triangle. Then the connection from the highest level gate to the second highest level gate is redundant for realizing the output of the lowest level gate if the second highest level gates has no output connection other than the one to the lowest level gate. In Fig. 3.3.6-1(a), the dash-lined connection is redundant. Conversely, if two gates are both immediate predecessors of another gate, adding a connection between the first two will not affect the output of the other gate if the gate to which the new connection feeds has no output connection other than the one to the third gate. In Fig. 3.3.6-1(b), gate I and gate J are immediate predecessors of gate K. The connection of gate I to gate J will not change the output of gate K. But the output of gate J after making the bold-line connection may be different



(a) The dashed connection is redundant



(b) The bold-line connection can be made without changing the output of gate K

Fig. 3.3.6-1 Triangular condition

from that before making the connection. In a similar manner, if three gates are all immediate predecessors of another gate, then adding one or two connections from some gates to the third gate will not affect the output of the lowest level gate. In Fig. 3.3.6-2, the networks in (a), (b), (c) and (d) realize the same function, but the outputs of gate K are not necessarily the same. Therefore, these different outputs of gate K can be used to compensate for error-components for other gates.

The procedures based on error-compensation can be briefly explained by the following four steps:

Step 1: Select a gate in the given network according to an appropriate order*. If all gates have been considered, then stop.

Step 2: Assume the selected gate is removed. Check whether the outputs of the network change or not. If the outputs do not change, then actually remove this gate and go to step 1. Otherwise go to next step.

Step 3: Construct the potential output table and go to step 4.

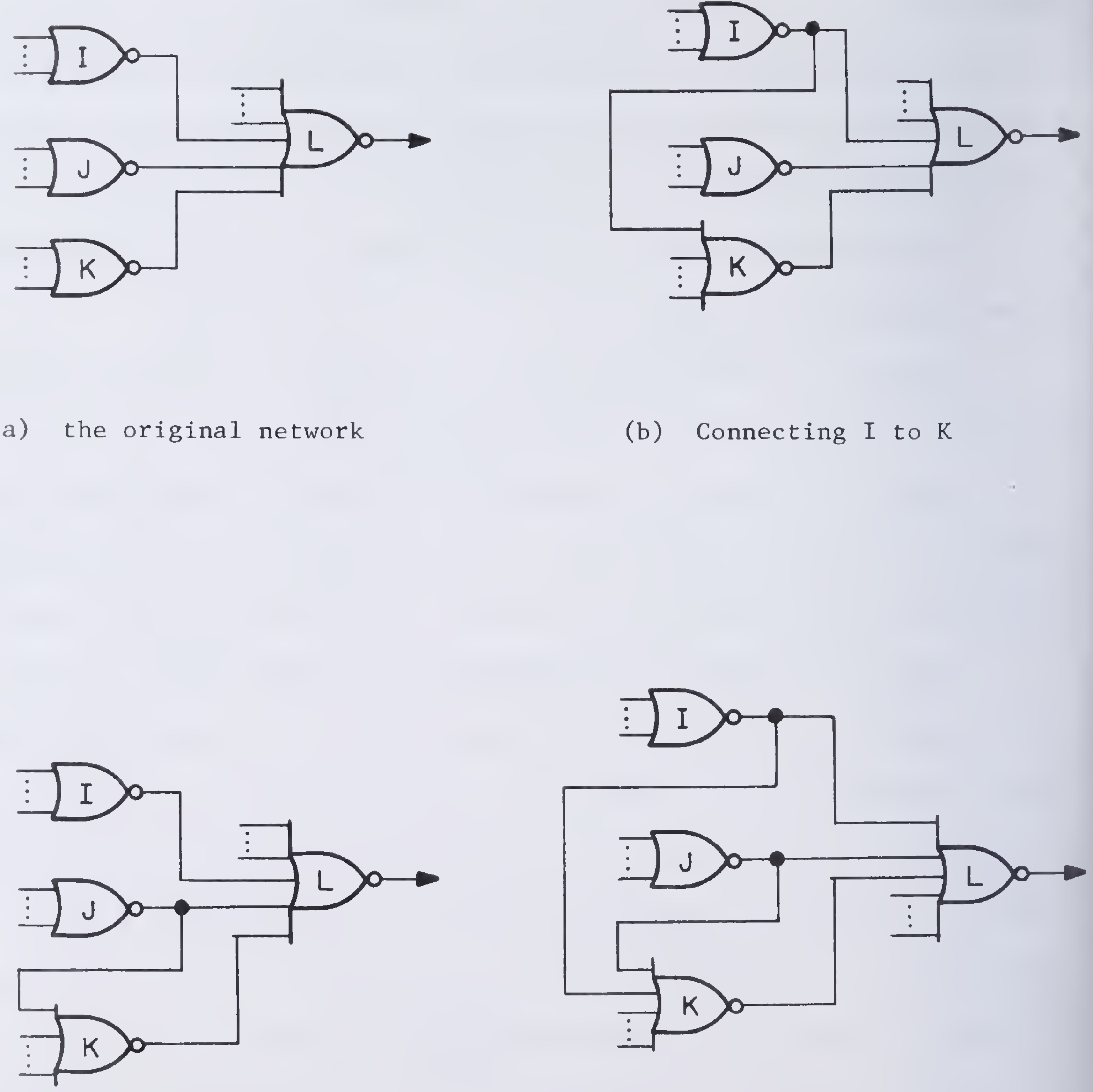
Step 4: Starting from the output gates, try to find some potential output functions which can compensate for the error-components found in step 2. If the errors can be compensated, then remove the selected gate and go to step 1. Otherwise, propagate the errors to the next higher level and repeat this step. If the errors have already been propagated to the highest level and still cannot be compensated, then go to step 1.

Step 4 is much more complicated than steps 1, 2 and 3. In step 4 six subprocedures are used to compensate for errors:

(1) Remove redundant connections.

(2) Substitution for input connections from external variables with error-components.

*Usually according to the number of 0-components a gate has.



(a) the original network

(b) Connecting I to K

(c) Connecting J to K

(d) Connecting I and J to K

Fig. 3.3.6-2 A more general example of triangular condition

(3) Substitution for input connections from gates with primary errors.

(4) Substitution for input connections by functions without error-components.

(5) Adding connections to compensate for 1-error components.

(6) Adding redundant connections from external variables.

Subprocedure (1) through subprocedure (4) aim at compensating for 0-error components while subprocedure (5) aims at compensating for 1-error components. Subprocedure (6) only loosens the requirements of the predecessors of some gate to make error-compensation at later steps easier.

The procedures based on error-compensation are much more complicated than any other transduction procedures. The details can be found in [15], [21].

The procedures based on error-compensation are inplemented in the transduction programs NETTRA-E1, NETTRA-E2 and NETTRA-E3. The essential difference among NETTRA-E1, -E2 and -E3 is that the central subroutines which realize the procedures based on error-compensation are called in different ways in NETTRA-E1, -E2 and -E3 [21].

3.3.7 Considerations of fan-in/fan-out restrictions and level restriction

In the previous sections, we do not consider the fan-in/fan-out restrictions and the level restriction in the transduction procedures. Since all IC logic families do have limits on the maximum number of fan-in and/or fan-out that a logic gate may have, the design of logic networks under the fan-in/fan-out restrictions is important. Besides, often we desire to design a "fast network", i.e., a network with short time delay between the inputs and the outputs, so the consideration of the maximum number of levels (which is proportional to the time delay) in a network as a restriction is required.

It was mentioned previously that the pruning procedures aim at removing redundant connections only. For any given network which is already fan-in/fan-out restricted and/or level-restricted, there will be no fan-in/fan-out problem or level problem to be generaged by applying the pruning procedures. But this is not true if the transduction procedures other than the pruning procedures are applied. All transduction procedures, except the pruning procedures and the procedure based on generalized gate substitution[*], are modified to take into consideration the fan-in/fan-out restrictions and the level restriction. We will discuss the modifications briefly, assuming that the given network is already fan-in/fan-out restricted and level restricted.

The following three cases will be discussed separately both for the fan-in/fan-out restrictions and for the level restriction, since all transduction procedures consist of one or more of these operations:

(a) Adding a connection from an external variable or a gate v_i to another gate v_j.

(b) Merging two gates v_i and v_j into a one gate v_{ij} by connecting a set of external variables and gates as input connections to gate v_{ij}.

(c) Substituting the current input connections or a subset of the current input connections of a gate by another set of external variables and the outputs of gates.

I Consideration of fan-in/fan-out restrictions

In case (a) the following conditions must be satisfied before adding the connection in order not to violate the fan-in/fan-out restrictions:

[*] The reason that this procedure is not modified is given in Chapter 5.

$$|IS(V_i)| < \text{FOX or FO and}$$
$$|IP(v_j)| < \text{FI},$$

where $|IS(v_i)|$ and $|IP(v_j)|$ are the current number of fan-out and fan-in of the external variable or gate v_i and the gate v_j, respectively. If neither of the above two conditions is satisfied, then the new connection cannot be made even if all other conditions except the above two are satisfied.

In case (b) each external variable or gate v which is going to be connected to gate v_{ij} as an input must satisfy

$|IS(v)| <$ FOX if v is an external variable

or

$|IS(v)| <$ FO (or FOO) if v is a gate (or an output gate)

in order not to violate the fan-out restrictions. The total number of the above v's must be less than or equal to FI in order not to violate the fan-in restriction. If we cannot find a set of v's such that the complement of the disjunction of f(v)'s belongs to $G_c(v_{ij})$ and the total number of v's is equal to or less than FI (described in section 3.3.3), then we cannot merge gates v_i and v_j.

Since gate v_{ij} must be connected to all immediate successors of gate v_i and gate v_j, we have to check whether $|IS(v_i) \cup IS(v_j)| \leq$ FO or not in order not to violate the fan-out restriction. If this condition is not satisfied then we cannot merge gates v_i and v_j.

In case (c) let us call the <u>substituting set</u> to be S' and the <u>substituted set</u> to be S. Also let the gate under consideration be v_i. Each external variable or gate v in the substituting set S' must satisfy the following conditions in order not to violate the fan-out restrictions:

$$|IS(v)| < FOX \text{ if } v \text{ is an external variable}$$

or

$$|IS(v)| < FO \text{ (or } FOO) \text{ if } v \text{ is a gate (or an output gate).}$$

Besides, the following condition must be satisfied in order not to violate the fan-in restriction at gate v_i:

$$|IP(v_i)| - |S| + |S'| \leq FI$$

The left hand side of the above inequality is the number of fan-in of gate v_i if the substitution is made. If the above inequality is not satisfied then we cannot make the substitution even if all other conditions are satisfied.

II Consideration of level restriction

Let the gate level of the output gate be 1[*], the gate level of the gate v be GLEVEL(v), and the maximum number of levels that the network may have be LEVLIM.

In case (a) if v_i is an external variable, then there will be no level problem caused by connecting v_i to v_j as an input of v_j. If v_i is a gate and $\mathrm{GLEVEL}(v_i) > \mathrm{GLEVEL}(v_j)$, then there will still be no level problem. But if v_i is a gate and $\mathrm{GLEVEL}(v_i) \leq \mathrm{GLEVEL}(v_j)$, then the following conditions must be satisfied in order not to violate the level restriction:

$$\mathrm{GLEVEL}(v_j) + \mathrm{DIST} \leq \mathrm{LEVLIM},$$

where DIST is the largest difference of gate levels between gate v_i and all predecessors of gate v_i. If the above condition is not satisfied, then the connection cannot be made.

[*] We may not be able to do so for every output gate in the multiple-output case since in this case the output of some output gate can also feed other gates.

In case (b), for every external variable v which is to be connected to gate v_{ij} as an input of v_{ij}, there will be no level problem. But the following conditions must be satisfied for a gate v which is going to be connected to gate v_{ij} in order not to violate the level restriction:

$$\text{DIST} + 1 + \max\ (\text{GLEVEL}(v_i), \text{GLEVEL}(v_j)) \leq \text{LEVLIM},$$

where DIST is the largest difference between gate v and all predecessors of gate v. Gates v_i and v_j cannot be merged if we cannot find a set S such that the conditions described in section 3.3.3 are satisfied and the above condition is satisfied by each gate in S.

In case (c) each gate in the substituting set must satisfy the same condition as case (a).

The modifications for fan-in/fan-out restrictions and level restriction are implemented in the transduction subroutines based on gate merging, gate substitution, connectable and disconnectable functions and error-compensation. Detailed explanations about the level restriction and the fan-in/fan-out restrictions are given in [12] and [9], [11], [38]*, respectively.

3.3.8 Assembly of the transduction programs for the NETTRA system

The transduction programs consist of five types of subroutines: the MAIN control subroutine, the I/O supporting subroutines, the subroutines for

*In [9], [11] and [38], the transduction programs modified for fan-in/fan-out restrictions are renamed as NETTRA-G1-FI-FO, -G2-FIFO, -G3-FIFO, -G4-FIFO, -E1-FIFO and -E2-FIFO.

finding initial networks, the subroutines for doing fan-in/fan-out restricted transformations and the subroutines for realizing the transduction procedures. In order to implement the NETTRA system, a new MAIN control subroutine and several new I/O supporting subroutines (the functions of these subroutines will be described in Chapter 4) are written to organize the subroutines for finding initial networks, for doing the transformations and for realizing the transduction procedures (except for NETTRA-E3) together.

Each type of transduction procedures (e.g., the transduction procedures based on gate merging and gate substitution constitute two different types.) are usually realized by more than one subroutine. Table 3.3.8-1 gives the names of the central transduction subroutines in the transduction programs implemented in the past for realizing the corresponding transduction procedures. For example, the central transduction subroutine in the transduction program NETTRA-G1 (the 5th row in Table 3.3.8-1) is named PRIIFF. It realizes the transduction procedures based on connectable and disconnectable functions.

The pruning procedures based on CSPF's are realized by the central subroutine PROCIP in the transduction program NETTRA-PG1. This type of pruning procedure has been found very efficient in removing redundant connections. Therefore they are included in many other transduction programs to remove redundant connections in a given network before applying more complex transductions. This is the reason why pruning appears in the transduction programs NETTRA-G1, -G2, -G3, -G4, -E1, -E2, and -E3 in Table 3.3.8-1.

Notice that the subroutines for implementing the transduction program NETTRA-E3 are not included in the NETTRA system. The transduction program NETTRA-E3 is a "multi-path" application of the error-compensation procedures

Table 3.3.8-1 The central transduction subroutines of the transduction programs implemented in the past for realizing the corresponding transduction procedures.

Transduction Program NETTRA-	Central Transduction Subroutine	Transduction Procedures Realized By The Central Subroutine
PG1	PROCIP	Pruning and gate substitution*
P1	RDTCNT	Pruning
P2	PROCI	Pruning
G1	PRIIFF	Connectable and disconnectable functions and Pruning
G2	PROCIV	
G3	GTMERG	gate merging and pruning
G4	PROCV	gate substitution and pruning
E1	PROCCE	error compensation and pruning
E2	PROCCE	
E3	PROCCE	

*A simple type of substitution procedures is also realized by this subroutine.

(see [21] for details). It needs 43K more core storage than the transduction program NETTRA-E2 and it usually takes much more time than NETTRA-E2.

Although the transduction programs (except NETTRA-E3) are all included and their names are not referred to separately in the NETTRA system, their symbolic representations (e.g. NETTRA-PG1, NETTRA-P1, etc.) are convenient for referring to or for classifying the transduction procedures they realize. Some mnemonic names for the transduction procedures are designated based on the original symbolic names of the transduction programs for the above purpose in setting up input data. This will be explained in more detail in [13].

4. Detailed Organization of the NETTRA System

In this chapter we will first introduce the functions of some important subroutines. Then we will provide the detailed organization of the control subroutine MAIN. Since the total program size of the NETTRA system is greater than the maximum main storage available for the IBM 360/75J, the overlay structure is used in programming the NETTRA system. We will also explain what the overlay structure is and how we programmed the NETTRA system using the IBM 360/75J computer's overlay structure facility.

4.1 Functions of Important Subroutines

In Fig. 2.1 we give the general flowchart for the NETTRA system. All subroutines included in the system are classified into the following five groups:

(1) Subroutine MAIN (will be introduced in section 4.2).

(2) Subroutines for realizing the initial network methods.

(3) Subroutines for realizing the fan-in/fan-out transformation.

(4) Subroutines for realizing the transduction procedures.

(5) I/O supporting subroutines.

The fan-in/fan-out transformations, some of the initial network methods and some of the transduction procedures are realized by more than one subroutine. Only the central subroutines that realize the initial network methods, the fan-in/fan-out transformations and the transduction procedures are described below:

Subroutines for realizing the initial network methods

BANDB: This is the central subroutine for realizing the initial network method based on the branch-and-bound algorithm (see section 3.1.3).

TANT: This is the central subroutine for realizing the initial network method based on Gimpel's algorithm (see section 3.1.5).

TISON: This is the central subroutine for realizing the initial network methods based on Tison's algorithm. The minimum sum is first found by this subroutine, and then the corresponding two-level or three-level network is constructed (see section 3.1.4).

TISLEV: This subroutine realizes the level-restricted initial network method by expanding the network obtained by Tison's method (see section 3.1.6). It will be applied only when both the fan-in/fan-out restrictions and level restriction exist.

THRLEV: This subroutine realizes the three-level network method for finding the initial networks (see section 3.1.2).

UNIVSA: This subroutine realizes the universal NOR network method for finding the initial networks (see section 3.1.1).

EXNT: This subroutine can "input" an initial network which may be obtained by any other than the above six methods.

Subroutine for realizing the fan-in/fan-out restricted transformations

JEFF: This is the central subroutine for realizing the fan-in/fan-out restricted network transformations (see section 3.2).

Subroutines for realizing the transduction procedures

MINI2*: This subroutine calculates the CSPF's of a given network and removes redundant connections, i.e., it realizes the pruning procedure based on CSPF's (see section 3.3.2).

*This is one of the two central subroutines for subroutine PROCIP which is mentioned in Chapter 3.

SUBSTI*: This subroutine realizes the transduction procedure based on simple substruction (see section 3.3.4).

RDTCNT: This subroutine realizes the pruning procedure based on OFMSPE (see section 3.3.2).

PROCI: This subroutine realizes the pruning procedure based on 1FMSPF (see section 3.3.2).

GTMERG: This subroutine realizes the transduction procedure based on gate merging (see section 3.3.3).

PROCV: This is the central subroutine for realizing the transduction procedures based on generalized gate substitution (see section 3.3.4).

PRIIFF: This is the central subroutine for realizing the transduction procedures based on connectable and disconnectable functions. This subroutine does not try to remove specific gates (see section 3.3.5 or see [1] for details).

PROCIV: This is the central subroutine for realizing the transduction procedures based on connectable and disconnectable functions. But this subroutine concentrates on removing specific gates (see section 3.3.5 or see [1] for details).

PROCCE: This is the central subroutine for realizing the transduction procedures based on error-compensation (see section 3.3.6).

I/O Supporting subroutines

INPUT: This subroutine reads the control sequence cards. The information specified on the control sequence cards decides the types of the initial network subroutines and the transduction subroutines to be applied.

SETEX: This subroutine sets up a truth table for n external variables when only uncomplemented external variables are permitted.

*This is the other central subroutine for subroutine PROCIP which is mentioned in Chapter 3.

PUSHIN: This subroutine sets up a truth table for n complemented external variables when both complemented and uncomplemented external variables are permitted.

SUBNET: This subroutine may be entered at three different points by a call to either SUBNET, UNNECE or PVALUE.

SUBNET generates detailed information on the network configuration. It calculates the level number of each gate, lists all immediate successors and immediate predecessors for each gate and it also calculates the successor matrix which indicates the existence or non-existence of a path from gate i to j.

UNNECE disconnects certain types of obviously unnecessary connections in the network.

PVALUE calculates the actual truth table for the entire network (except external variables).

OUTPUT: This subroutine may be entered at five different points by a call to either OUTPUT, PAGE, LINE, TRUTH or CKT.

OUTPUT assigns mnemonic names to external variables and gates for the purpose of achieving a readable print-out.

PAGE ejects one page on the printer.

LINE skips a specified number of lines on the print-out sheet. The number is specified by the argument in the call (e.g., "CALL LINE(5)" skips 5 lines).

TRUTH prints out the truth table of the network currently stored as INC$MX.

CKT prints out the network itself.

SUMARY: This subroutine will prepare a summary for the result of each problem. The summary contains the best cost obtained, the computation time spent and the given problem specifications (the number of external variables, the number of output functions, the maximum fan-ins, fan-outs and the maximum levels).

PUNCAD: This subroutine will punch on cards the information of the current network configuration when the specified time is to expire but the best network is not yet obtained.

PARMP: This subroutine will punch cards for the unfinished jobs. The cards contain the information of the original problem specifications, the output functions, the control sequence and any other information for continuing the unfinished jobs.

4.2 Control Subroutine MAIN

Since the NETTRA system combines all of the transduction programs which were developed previously into one big program, it has the ability to do everything that each transduction program can do. Besides, the NETTRA system gives the user more flexibility in designing networks. The user can specify the type, the order and the number of times that some initial network subroutines or some transduction subroutines will be applied. The NETTRA system also has the facilities to treat the intermediate results when specified computation time is to expire; the user can continually run the program next time without losing anything on the unfinished problem. The control subroutine MAIN for the NETTRA system, hence, is very much complicated.

Any given problem, if there exists only fan-in/fan-out restrictions, is dealt with by the subroutine MAIN according to the general flowchart shown in Fig. 4.2-1. In Fig. 4.2-1 all decision blocks and execution blocks are labeled with integers. We explain these blocks in detail below:

Block 1: Check whether all problems are solved or not (more than one problem can be submitted at each run). If all problems are solved, then stop. Otherwise go to block 2.

Block 2: Read in a new problem. Initialize some flags and parameters for further processing. Go to block 3.

Block 3: Check whether all specified initial network methods are applied or not (this means that for any problem we can start from initial networks derived by different methods and then apply the same non-fan-in/non-fan-out restricted transduction -- fan-in/fan-out restricted transformation -- fan-in/fan-out restricted transduction sequence to get different results). If this is true, then go back to block 1; otherwise go to block 4.

Block 4: Select the next initial network method and call the corresponding subroutine and go to block 5.

Block 5: Check whether all specified non-fan-in/non-fan-out restricted transduction subroutines are applied or not. If this is true, then go to block 8. Otherwise go to block 6.

Block 6: Select the next transduction procedure and call the corresponding subroutine without considering the fan-in/fan-out restrictions and go to block 7.

Block 7: Check whether or not the selected transduction subroutine (in block 6) has been called the specified number of times. If this is true, then go back to block 5. If this is not true but the network cost is not improved after applying block 6, then we can also terminate the application of block 6 and go back to block 5. Otherwise go back to block 6.

Block 8: This block is reached from block 5. The fan-in/fan-out restricted transformation subroutine JEFF is applied to transform the network obtained in block 5 into a fan-in/fan-out restricted network. Go to block

Block 9: Check whether all specified fan-in/fan-out restricted transduction subroutines are applied or not. If this is true, then go to block 12. Otherwise go to block 10.

Block 10: Select the next transduction procedure and call the corresponding subroutine considering the fan-in/fan-out restrictions. Go to block 11.

Block 11: Check whether or not the selected transduction subroutine (in block 10) has been called the specified number of times. If this is true, then go to block 9. If this is not true but the network cost is not improved, then go to block 9. Otherwise go to block 10 to call the selected transduction subroutine again.

Block 12: This block can only be reached from block 9. It checks whether or not the loop consisting of non-fan-in/non-fan-out restricted transduction -- fan-in/fan-out restricted transformation -- fan-in/fan-out restricted transduction has been applied the specified number of times. If this is true, then go back to block 3. This outer loop can also be terminated if there is no improvement in the cost. Otherwise go back to block 5.

The following is an example which may facilitate the understanding of Fig. 4.2-1.

Example 4.2-1 -- For a given function f, let us apply the following initial network subroutines and the transduction subroutines:

Initial network subroutines: UNIVSA, THRLEV, BANDB

non-fan-in/non-fan-out restricted transduction subroutines:

MINI2	2 times
PRIIFF	2 times
GTMERG	3 times
PROCCE	2 times

fan-in/fan-out restricted transduction subroutines:

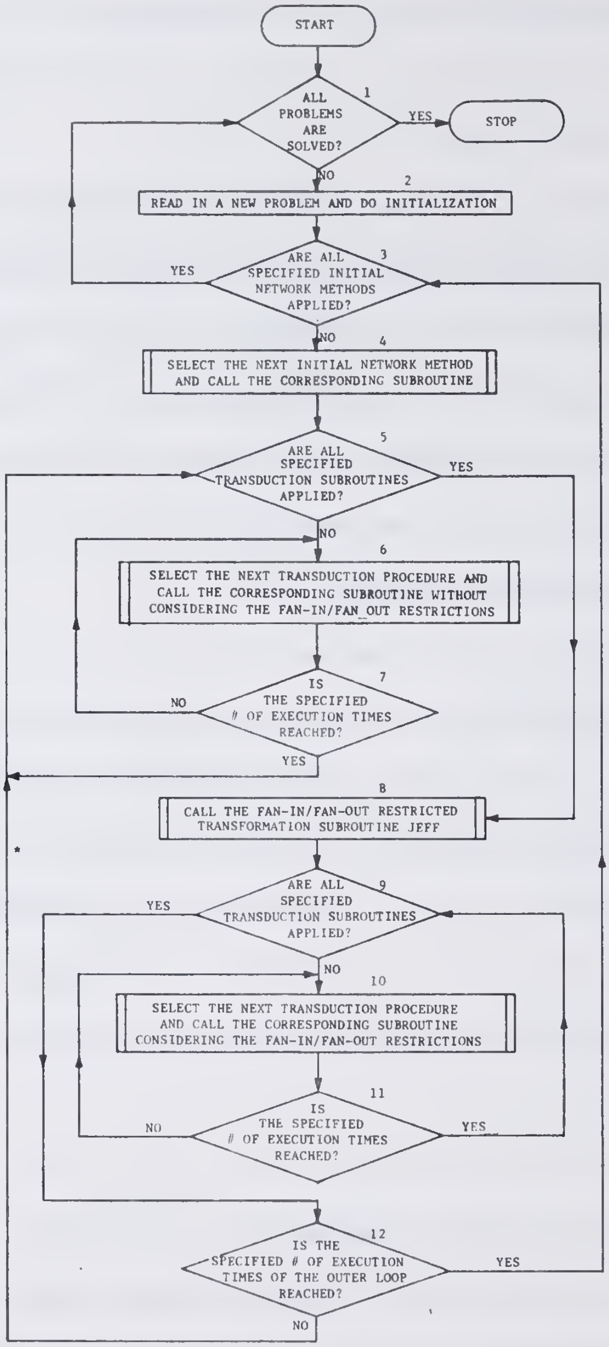

START
1
ALL PROBLEMS ARE SOLVED?
YES
STOP
NO
2
READ IN A NEW PROBLEM AND DO INITIALIZATION
3
ARE ALL SPECIFIED INITIAL NETWORK METHODS APPLIED?
YES
NO
4
SELECT THE NEXT INITIAL NETWORK METHOD AND CALL THE CORRESPONDING SUBROUTINE
5
ARE ALL SPECIFIED TRANSDUCTION SUBROUTINES APPLIED?
YES
NO
6
SELECT THE NEXT TRANSDUCTION PROCEDURE AND CALL THE CORRESPONDING SUBROUTINE WITHOUT CONSIDERING THE FAN-IN/FAN OUT RESTRICTIONS
7
IS THE SPECIFIED # OF EXECUTION TIMES REACHED?
NO
YES
8
CALL THE FAN-IN/FAN-OUT RESTRICTED TRANSFORMATION SUBROUTINE JEFF
*
9
ARE ALL SPECIFIED TRANSDUCTION SUBROUTINES APPLIED?
YES
NO
10
SELECT THE NEXT TRANSDUCTION PROCEDURE AND CALL THE CORRESPONDING SUBROUTINE CONSIDERING THE FAN-IN/FAN-OUT RESTRICTIONS
11
IS THE SPECIFIED # OF EXECUTION TIMES REACHED?
NO
YES
12
IS THE SPECIFIED # OF EXECUTION TIMES OF THE OUTER LOOP REACHED?
YES
NO

Fig. 4.2-1 General flowchart for the subroutine MAIN when fan-in/fan-out restrions are considered.

[*] The execution of these loops can also be terminated whenever there is no further improvement in cost.

GTMERG 3 times

PROCCE 2 times

and the loop consisting of the non-fan-in/non-fan-out restricted transduction -- fan-in/fan-out restricted transformation -- fan-in/fan-out restricted transduction is to be executed 3 times.

The initial network subroutine UNIVSA will first be applied, and the subroutines MINI2, PRIIFF, GTMERG and PROCCE will be called one by one without considering the fan-in/fan-out restrictions. Each subroutine will be called as many times as specified, but its application will be stopped if the cost cannot be improved. The fan-in/fan-out restricted transformation subroutine JEFF is then called to transform the network into fan-in/fan-out restricted form. The transduction subroutines GTMERG and PROCCE will then be called as many times as specified, but its application will be stopped if the cost cannot be improved. The loop consisting of the non-fan/non-fan-out restricted transduction -- fan-in/fan-out restricted transformation -- fan-in/fan-out restricted transduction will be executed three times as specified. But if there is no improvement in cost after the first or second iteration, then it will be terminated. The initial network subroutines THRLEV and BANDB will then be applied one by one and the same loop consisting of non-fan-in/non-fan-out restricted transduction -- fan-in/fan-out restricted transformation -- fan-in/fan-out restricted transduction will be followed as before to get near-optimal fan-in/fan-out restricted networks.

For convenience, we will, from now on call a sequence of initial network subroutines, non-fan-in/non-fan-out restricted transduction subroutines, fan-in/fan-out restricted transformation subroutine, fan-in/fan-out restricted transduction subroutines, a control sequence to mean that the processing for the given problem is controlled by this sequence. We will also call a sequence of non-fan-in/non-fan-out restricted transduction subroutines, fan-in/fan-out restricted transformation subroutine, fan-in/fan-out restricted transduction subroutines, a TT-sequence (Transduction and Transformation sequence). This means that each control sequence is composed of one or more initial network subroutines and a TT-sequence.

In the case that both fan-in/fan-out restrictions and level restriction are imposed, then subroutine MAIN will deal with design problems following the general flowchart shown in Fig. 4.2-2. The detailed explanation of Fig. 4.2-2 is given below.

Block 1: Check whether all given design problems are solved or not. If not, then go to block 2; otherwise stop.

Block 2: Read in a new design problem and initialize some flags and parameters for further processing.

Block 3: Let LEVLIM be LREST, where LREST is the given maximum number of levels to be restricted and LEVLIM be the maximum number of levels to be used in finding a feasible network and LEVLIM $\geq$ LREST (this will be explained later). Go to block 4.

Block 4: Call the initial network subroutine TISLEV to get a network whose number of levels is less than or equal to LEVLIM. This initial network may or may not be fan-in/fan-out restricted. Go to block 5.

Block 5 through Block 9: Call the transduction subroutines SUBSTI, GTMERG, PRIIFF, PROCIV and PROCCE one by one considering both the fan-in/fan-out restrictions and the level restriction. In each block, each transduction subroutine will be repeatedly called until there is no further improvement in cost.

Block 10: Check whether the number of levels (denoted by LEVM) of the network obtained so far is less than or equal to the given restriction LREST or not. If this is true (the network is level-restricted), then go to block 12. Otherwise go to block 11.

Block 11: Check whether the corresponding initial network is fan-in/fan-out restricted or not. If it is not, then stop because no feasible solutions can be obtained. Otherwise go to block 14.

Block 12: This block is reached from block 10. Check whether the network obtained so far is fan-in/fan-out restricted or not. If it is, then go to block 13, since a feasible network is obtained. Otherwise go to block 14.

Block 13: Output the feasible network and go back to block 1.

Block 14: When this block is reached, the network obtained at this point must belong to one of the following two cases:

(1) It is level-restricted but not fan-in/fan-out restricted.

(2) It is not level-restricted and its corresponding initial network is not fan-in/fan-out restricted.

For both cases, the "relaxation problem" is considered, i.e., we increase LEVLIM by 1 and go back to block 3 to find another initial network with a higher level limit. This new initial network apparently will not be level-restricted, but it may have less fan-in/fan-out problems than those initial networks with the lower level limit. Starting from this new initial network, the number of levels of the network may be reduced after applying the transduction procedures.

As is mentioned in Chapter 2, there may not exist any network which satisfies both the level restriction and the fan-in/fan-out restrictions. The general flowchart shown in Fig. 4.2-2 does not guarantee that a feasible network can be found even if there do exist optimal networks. But the statistics show that the results obtained by this approach are reasonably good (see

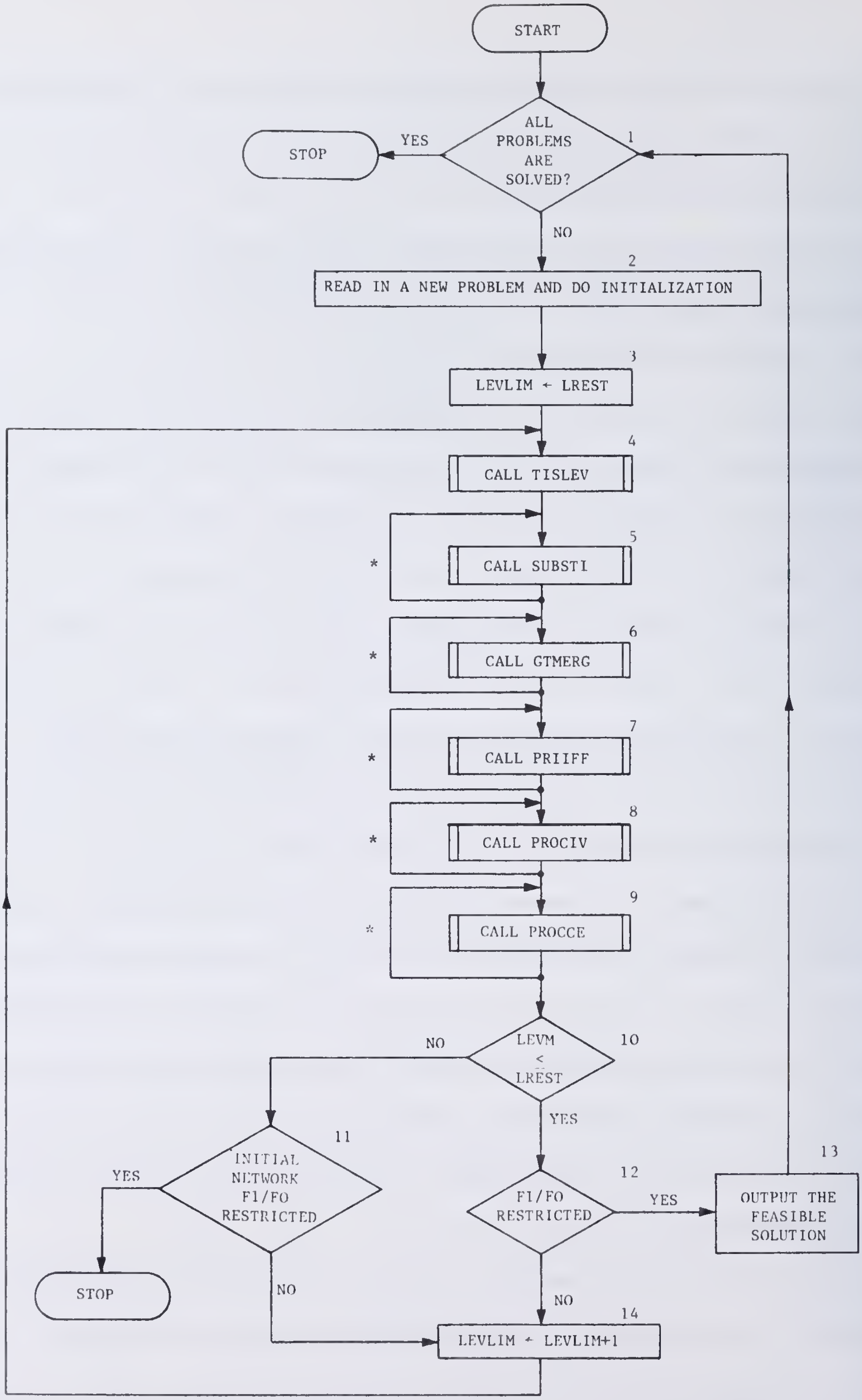


Fig. 4.2-2 General flowchart for the subroutine MAIN when both fan-in/fan-out restrictions and level restriction are considered.

Chapter 5 or see [12] for more details).

The detailed flowchart for the subroutine MAIN is shown in Fig. 4.2-3(a) through Fig. 4.2-3(f). This detailed flowchart will facilitate those readers who are interested in how the subroutine MAIN is actually programmed. In Fig. 4.2-3 one or more steps are grouped as a block (shown in each dashed rectangle), and the details for each block is explained below. Block 1 through block 7 (Fig. 4.2-3(a)) are for initialization. Block 8 through block 20 (Fig. 4.2-3(b)) are for finding the initial networks. Block 21 through block 29 (Fig.4.2-3(c)) are for applying the transduction procedures without considering fan-in/fan-out restrictions and level restriction. Block 30 through block 33 (Fig. 4.2-3(d)) are for applying the fan-in/fan-out restricted transformations. Block 34 through block 43 (Fig. 4.2-3(3)) are for applying the transduction procedures considering fan-in/fan-out restrictions (and level restriction). Block 44 (Fig. 4.2-3(f)) is for the control flow for considering the level restriction.

Block 1: Set the flag SIGNAL=0 and then read a "specification card". SIGNAL is used to tell whether the data to be read in belongs to the first problem set or not. Initially SIGNAL is set to zero; this means that the data to be read in belongs to the first problem set in this run. SIGNAL will be set and kept as 1 after the first problem has been read in. Six parameters are contained in the "specification card". The "specification card" is used to tell what kinds of input data cards are needed for each problem set. The meaning of those parameters will be explained later.

Block 2: Reset MSIIME, which is used to test whether the specified computation time limit is to expire or not. Check whether the heading* card should be read in or not depending on the values of HEAD and SIGNAL. If the current

*The heading card usually tells what problem is going to be solved.

problem is not the first problem and HEAD=1[*], then read the heading card. Go to block 3.

Block 3: Depending on the values of PARM and SIGNAL, check whether the parameter card should be read in or not. Twelve parameters are contained in the parameter card. The brief meaning of each parameter is introduced below:

N: the number of external variables.

M: the number of outputs (or the number of given functions).

A: cost coefficient for the number of gates.

B: Cost coefficient for the number of connections.

UC: this parameter indicates the type of external variables permitted -- only uncomplemented or both complemented and uncomplemented.

TFI: maximum number of fan-in for gates.

TFO: maximum number of fan-out for gates (not output gates).

TFOX: maximum number of fan-out for external variables.

TFOO: maximum number of fan-out for output gates.

LREST: maximum number of levels.

POPT1: this parameter indicates whether the detailed processes of each transduction probram are to be printed or not.

POPT2: this parameter indicates whether the detailed processes of the initial network program TISLEV are to be printed or not.

RERUN: It indicates whether the current problem is executed for the first time or it is not finished last time and will be continued this time. Normally RERUN=0.[†]

[*] This is the case that each of the following problems has its own heading card. HEAD=0 implies that the following proglems are the same as the first one except possible difference in some restrictions.

[†] For the sake of simplicity, we do not include the checks for rerunning the unfinished job in the flowchart. But when RERUN=1 then we will start from the point where the job was stopped last time.

NEPMAX: This parameter denotes the maximum number of error components permitted in the transduction procedures based on error-compensation. Usually it is not specified and will be set to value 2^{N-1} internally.

IF SIGNAL=1 and PARM=0, then no parameter card is needed for the current problem set. Go to block 4. Otherwise read in the parameter card and set default values for those parameters which are not specified by the users.

Block 4: Depending on the values of SIGNAL and SEQC, check whether or not the control sequence card(s) should be read in. If SIGNAL=1 and SEQC=0, then no control sequence card is needed for the current problem; go to block 5. Otherwise read in the control sequence card(s), encode the information into decimal numbers and store these numbers in proper arrays; go to block 5.

Block 5: Depending on the values of SIGNAL and FUNC, check whether or not the output function card(s) should be read in. If SIGNAL=1 and FUNC=0, then no output function card is needed; go to block 6. Otherwise read in the output function cards.

Block 6: Set SIGNAL=1 and select an initial network subroutine specified in the control sequence card(s) (this is equivalent to LIJ←LIJ+1, where LIJ is 0 initially). If all initial network subroutines have been applied, then go back to block 2 to read another problem set; otherwise go to block 7.

Block 7: Set information for making a summary at the end of this problem, set the truth table for external variables, and set initial values for some parameters. Now it is ready to start execution of the current problem.

Block 8: Set KKKK=INITYP (LIJ), where LIJ ranges from 1 through INTTMX which is the maximum number of different initial network methods to be applied. KKKK ranges from 1 through 6. Go to block 9.

<u>Block 9</u>: If KKKK=1, then call PUSHIN and BANDB to find an initial network by the branch-and-bound method. Go to block 16.

<u>Block 10</u>: If KKKK=2, then call NORNET to find an initial network by Gimpel's algorithm. Go to block 15.

<u>Block 11</u>: If KKKK= 3, then call THRLEV to find an initial network by the three-level network method. Go to block 15.

<u>Block 12</u>: If KKKK=4, then call UNIVSA to find an initial network by the universal NOR network method. Go to block 15.

<u>Block 13</u>: If KKKK=5, then call TISON to get a two-level or three-level initial network by Tison's method. If there exist level restriction and fan-in/fan-out restrictions, then call TISLEV to get a level-restricted network based on the two-level or three-level network just obtained. Call SETEX and then go to block 15.

<u>Block 14</u>: If KKKK=6, then call EXNT to read in the configuration of a network which may be found by any other method. Go to block 15.

<u>Block 15</u>: Call PUSHIN and go to block 16.

<u>Block 16</u>: Update the values of NR and NRN2. NR is the total number of gates and external variables. NRN2 is the size (number of rows times number of columns) of the truth table for all gates and external variables. Go to block 17.

<u>Block 17</u>: Call OUTPUT, SUBNET and PVAUE to set up the network configuration for the network configuration for the initial network. Go to block 18.

<u>Block 18</u>: Call TRUTH and CKT to print out the truth table and the network configuration for the initial network. Go to block 19.

<u>Block 19</u>: Update the value of MSTIME. Go to block 20.

<u>Block 20</u>: Check whether or not the specified computation time limit for the current problem is to expire. If not, then go to block 21 to continue the

execution; otherwise punch the intermediate results for the current problem and go back to block 2.

Block 21: Set an initial value for BSTCST (200000 is used) which keeps the best cost of the networks during the processes. Set FI=FO=FOX=FOO=100. FI, FO, FOX and FOO will be used as the fan-in/fan-out restrictions. Also set ITER=0, where ITER is used to count the number of times that the TT-sequence is executed. Go to block 22.

Block 22: Increase ITER by one and check whether ITER is greater than ITRMAX or not, where ITRMAX is the number of times that the specified TT-sequence is to be executed. GDCST (used to store the best cost found after each application of the TT-sequence) and NEWCST (used to store the cost found after each application of the transduction subroutine or the transformation subroutine) are initiated to the value COST which is the cost of the initial network. Go to block 23.

Block 23: Check if BSTCST $\geq$ GDCST or not. If this is true, then BSTCST is changed to GDCST (this means that a better network is obtained), store the network configuration of the better network for later use, and go to block 24. Otherwise go to block 24.

Block 24: Call UNNECE to remove some obviously redundant connections from the initial network. Go to block 25.

Block 25: Decode the array SEQE which contains the information about the control sequence. The type of the transduction subroutines (TSDTYP) and the number of execution times (ST2) are calculated in this block. If all specified transduction subroutines have been applied, go to block 30; otherwise go to block 26.

Block 26: Set TMPCST=NEWCST, where TMPCST is used to remember the best cost after the selected transduction subroutines are called in block 27 ST2 times. Go to block 27.

Block 27: Call the selected subroutine according to TSDTYP, and go to block 28.

Block 28: Check whether the specified computation time limit is to expire or not. If it is, then punch the intermediate results and go to block 2. Otherwise go to block 29.

Block 29: Check whether the selected transduction subroutine has been executed ST2 times or not. If so, then print the truth table and the network configuration of the network just found and go to block 25. Otherwise, check whether NEWCST < TMPCST or not, where NEWCST is the cost of the network just obtained. If it is true, then go to block 26. Otherwise go to block 25.

Block 30: This block is reached when all specified non-fan-in/non-fan-out restricted transduction subroutines have been applied. Set FI=TFI,FO=TFO, FOX=TFOX and FOO=TFOO and go to block 31.

Block 31: Call JEFF to transform the network into fan-in/fan-out restricted form. Go to block 32.

Block 32: Print the truth table and the network configuration for the network just obtained. Go to block 33.

Block 33: GDCST is then set to NEWCST, the cost of the network obtained in block 31. Check whether or not the specified computation time limit is to expire. If so, then punch the intermediate results and go to block 2. Otherwise go to block 34. It is now ready to do the fan-in/fan-out restricted transduction.

Block 34: Decode the array SEQE to find the type of the transduction subroutine (FIFOTP) and the number of execution times (ST4). If all specified transduction subroutines have been applied, then go to block 42; otherwise go to block 35.

Block 35: Set OLDCST to the value NEWCST, where OLDCST is used to store the network cost each time a "better network" (with lower cost) is obtained after applying the transduction subroutines. Go to block 36.

Block 36: Call the selected transduction subroutine and go to block 37.

Block 37: Print the truth table and the network configuration and then go to block 38.

Block 38: Check whether or not the specified computation time is to expire. If so, then punch the intermediate results and go to block 2. Otherwise go to block 39.

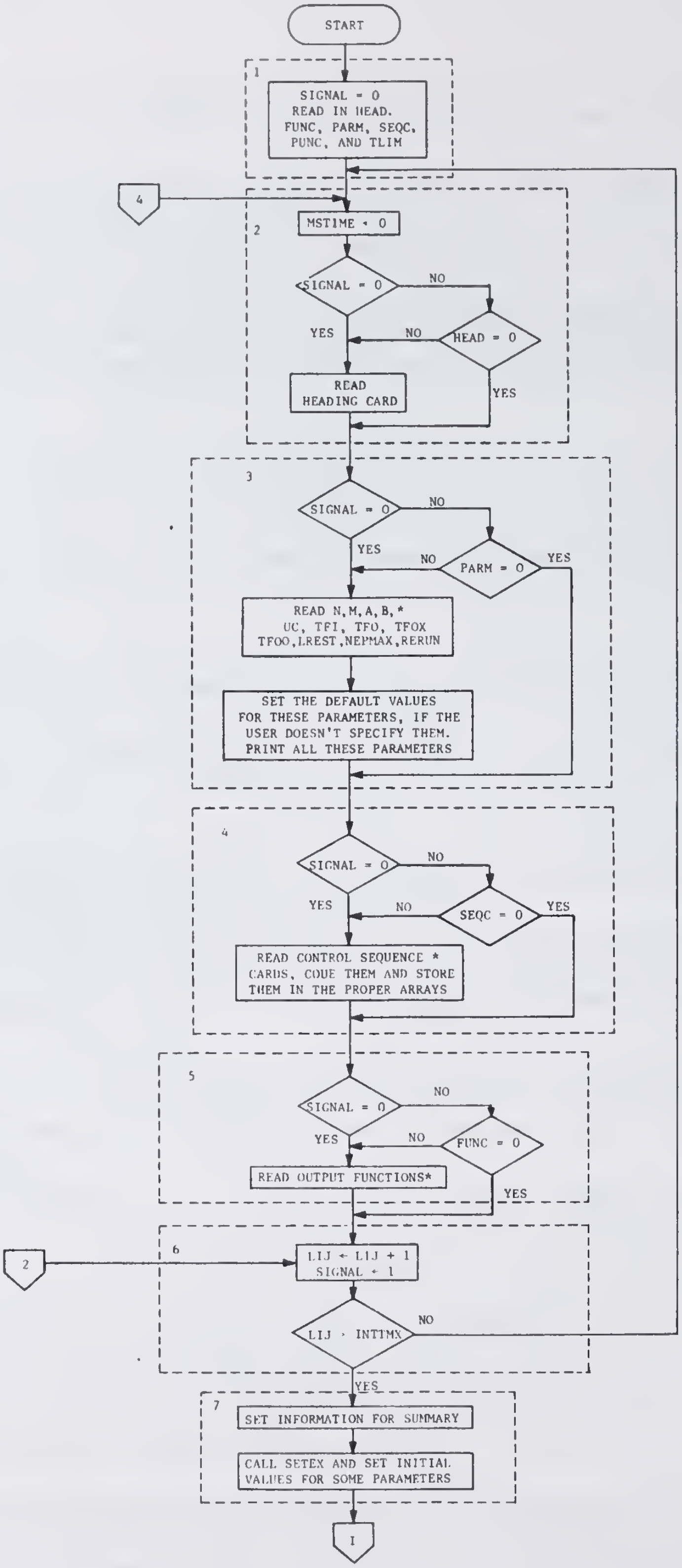
Block 39: Check whether or not the selected transduction subroutine has been executed ST4 times or not. If it is, then go to block 41. Otherwise go to block 40.

Block 40: Check whether NEWCST < OLDCST or not. If this is true, then a network with better cost is obtained. Go to block 35. Otherwise go to block 41.

Block 41: Check whether NEWCST < GDCST or not. If it is, then replace GDCST by NEWCST and go to block 34. Otherwise go to block 34.

Block 42: This block is only reached from block 34. It checks whether the TT-sequence has been applied ITRMAX times or not. If ITRMAX=99 (this means that the TT-sequence will be repeatedly applied until there is no further improvement in cost), then go to block 44. Otherwise check wither or not ITER, the number of times that the TT-sequence is executed, is less than ITRMAX. If it is, then go to block 23. Otherwise check whether the specified control sequence is for the fan-in/fan-out restricted and level-restricted problems. If it is, then go to block 45. Otherwise replace BSTCST by GDCST and store the corresponding network configuration, and go to block 6.

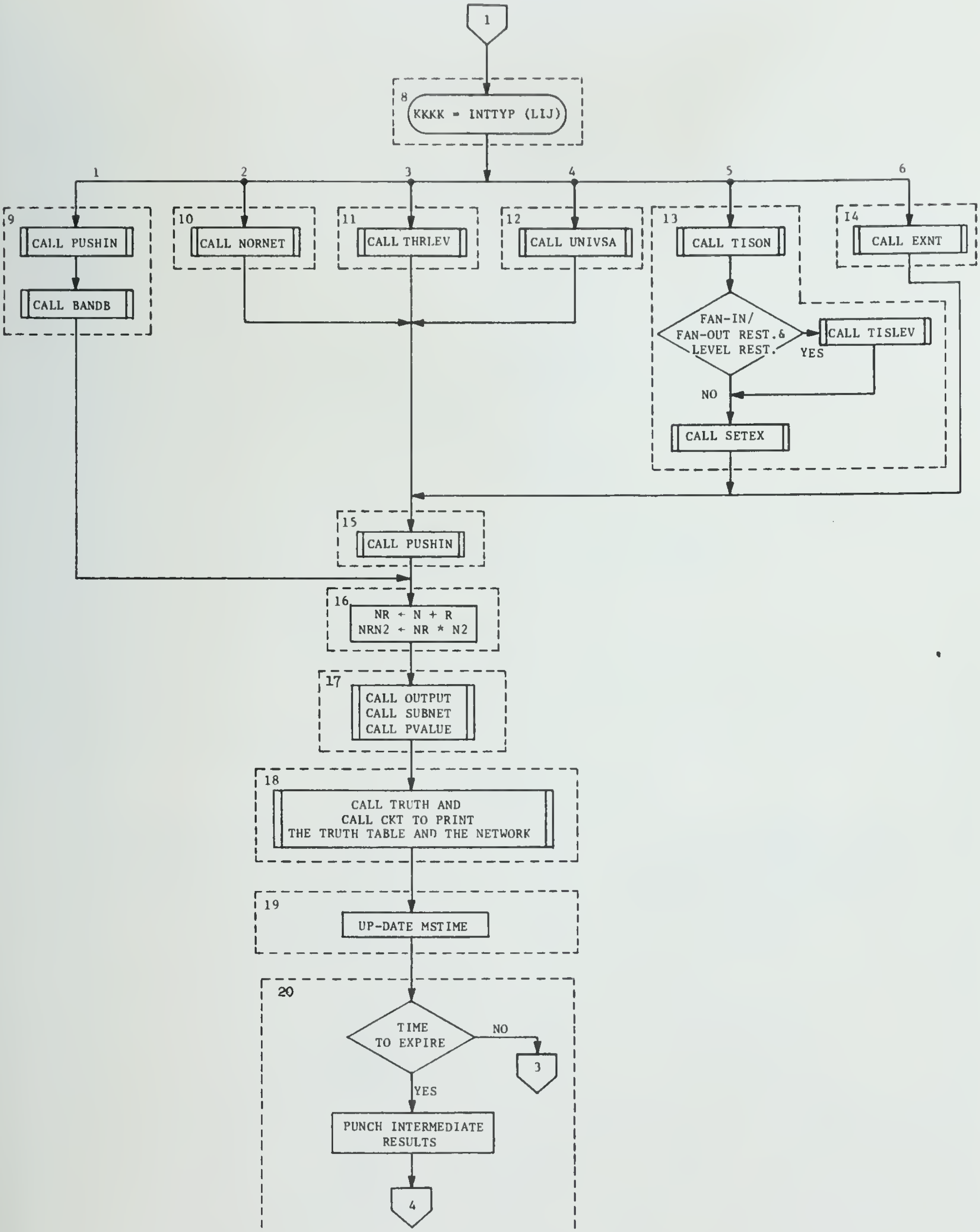
Block 43: This block can only be reached from blocks 2, 3, 4 and 5 when data cards are needed but the end of a file is encountered. At this point, the



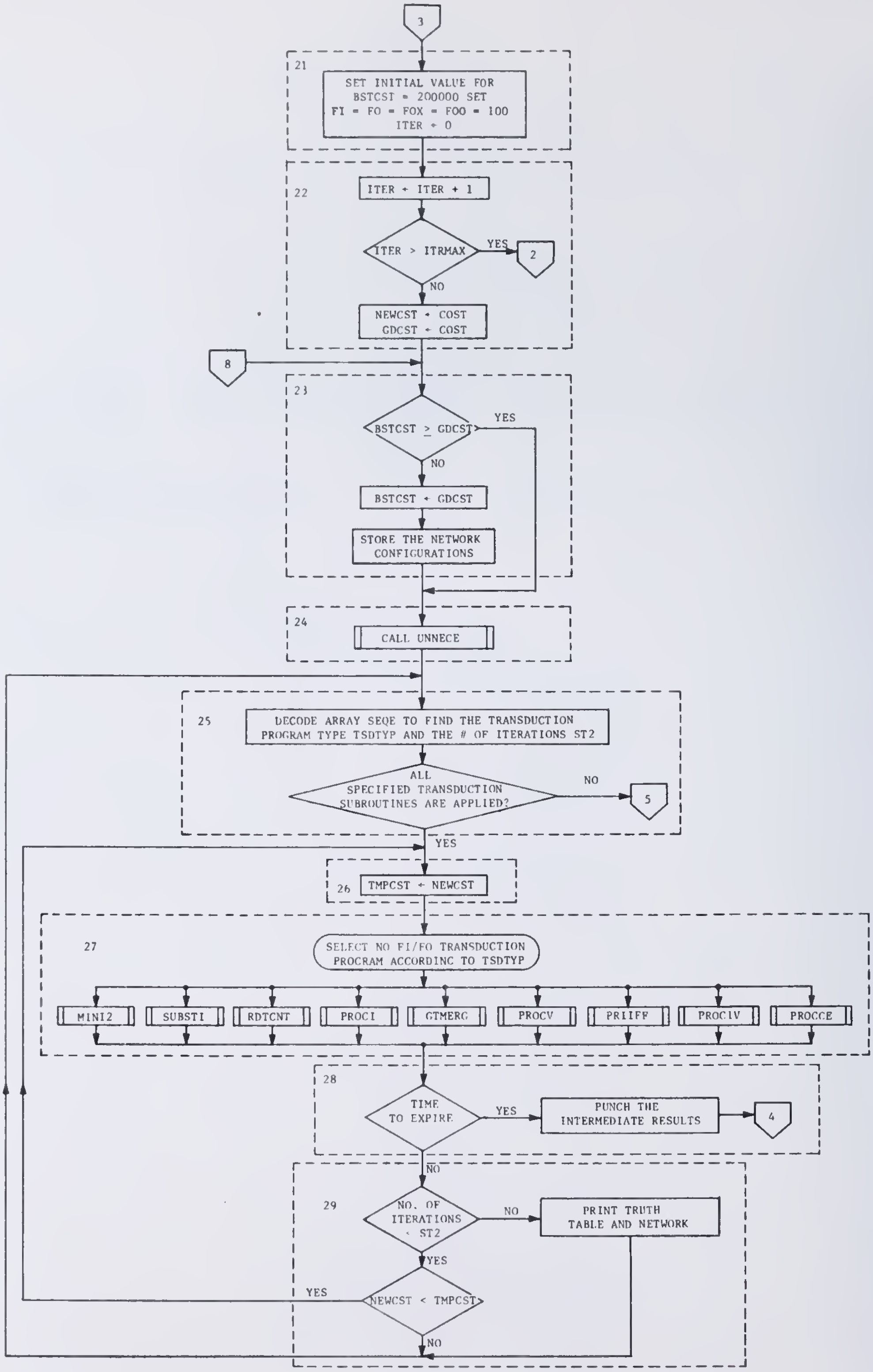
(a) Initialization

Fig. 4.2-3 Detailed flowchart for the control subroutine MAIN.

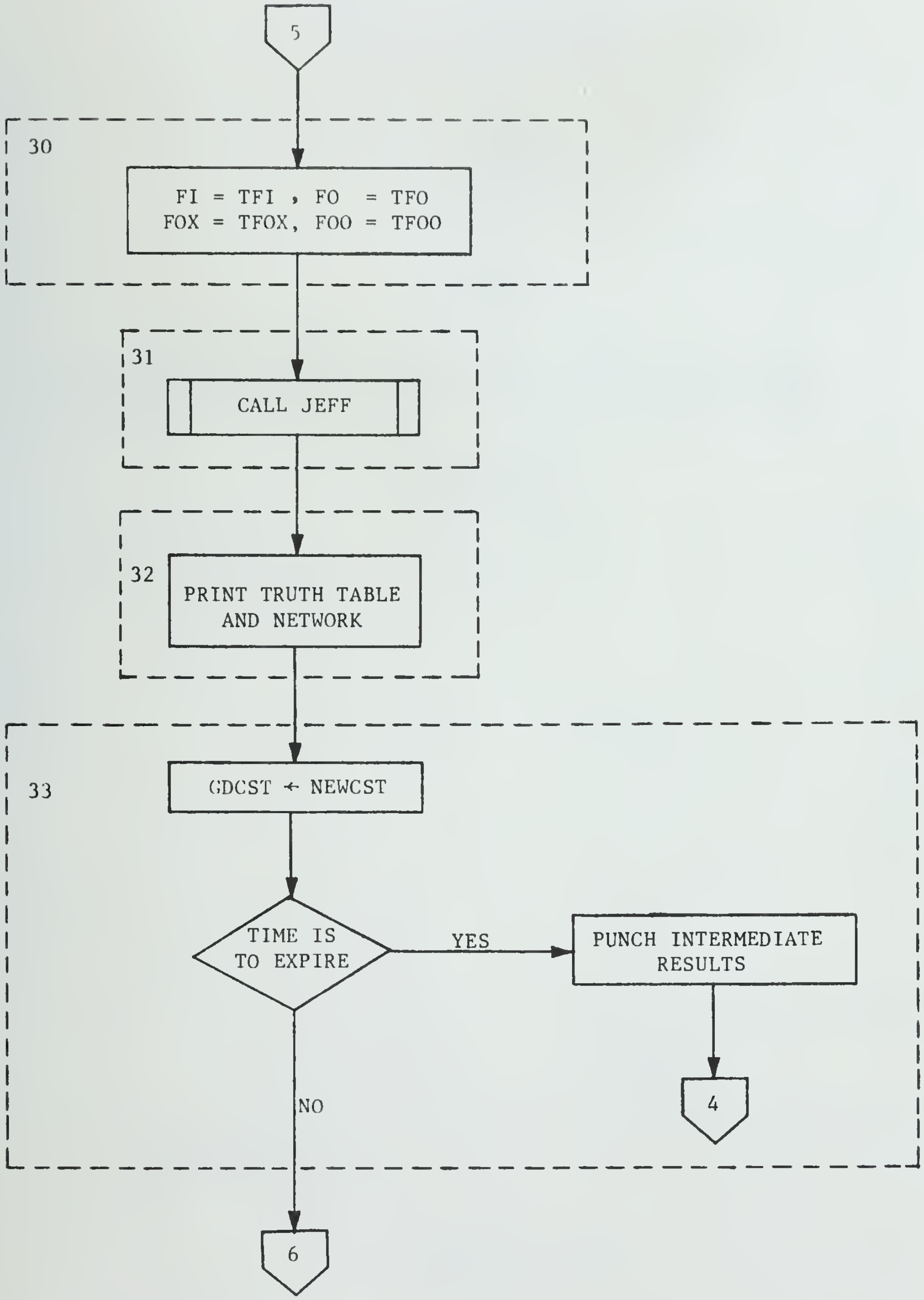
*If there is no data card then it will jump to 9 to terminate the problem abnormally.



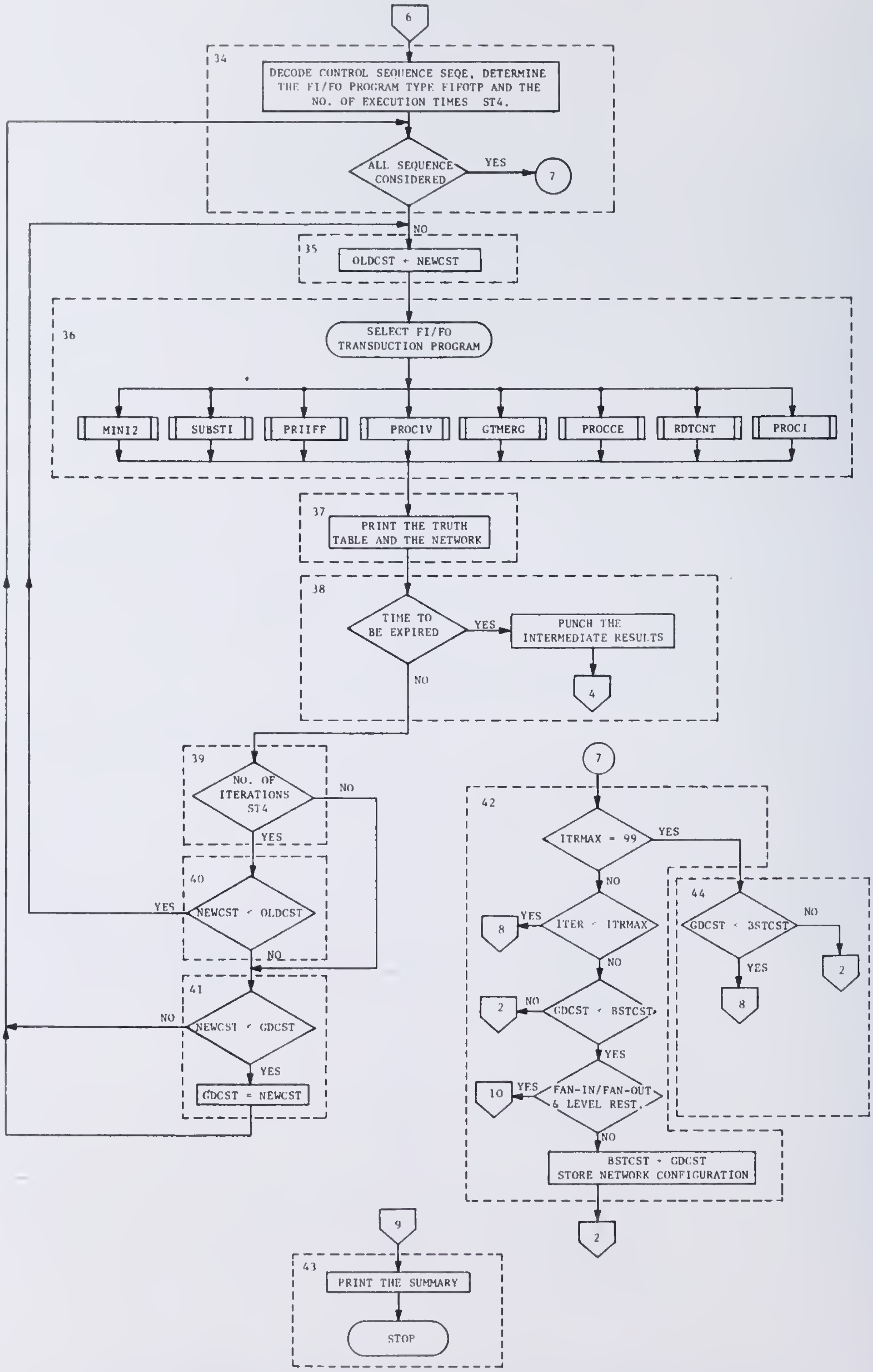
(b) Initial network step.


3
21
SET INITIAL VALUE FOR BSTCST = 200000 SET FI = FO = FOX = FOO = 100 ITER ← 0
22
ITER ← ITER + 1
ITER > ITRMAX
YES
2
NO
NEWCST ← COST
GDCST ← COST
8
23
BSTCST ≥ GDCST
YES
NO
BSTCST ← GDCST
STORE THE NETWORK CONFIGURATIONS
24
CALL UNNECE
25
DECODE ARRAY SEQE TO FIND THE TRANSDUCTION PROGRAM TYPE TSDTYP AND THE # OF ITERATIONS ST2
ALL SPECIFIED TRANSDUCTION SUBROUTINES ARE APPLIED?
NO
5
YES
26
TMPCST ← NEWCST
27
SELECT NO FI/FO TRANSDUCTION PROCRAM ACCORDINC TO TSDTYP
MINI2
SUBSTI
RDTCNT
PROCI
GTMERG
PROCV
PRIIFF
PROCIV
PROCCE
28
TIME TO EXPIRE
YES
PUNCH THE INTERMEDIATE RESULTS
4
NO
29
NO. OF ITERATIONS < ST2
NO
PRINT TRUTH TABLE AND NETWORK
YES
YES
NEWCST < TMPCST
NO

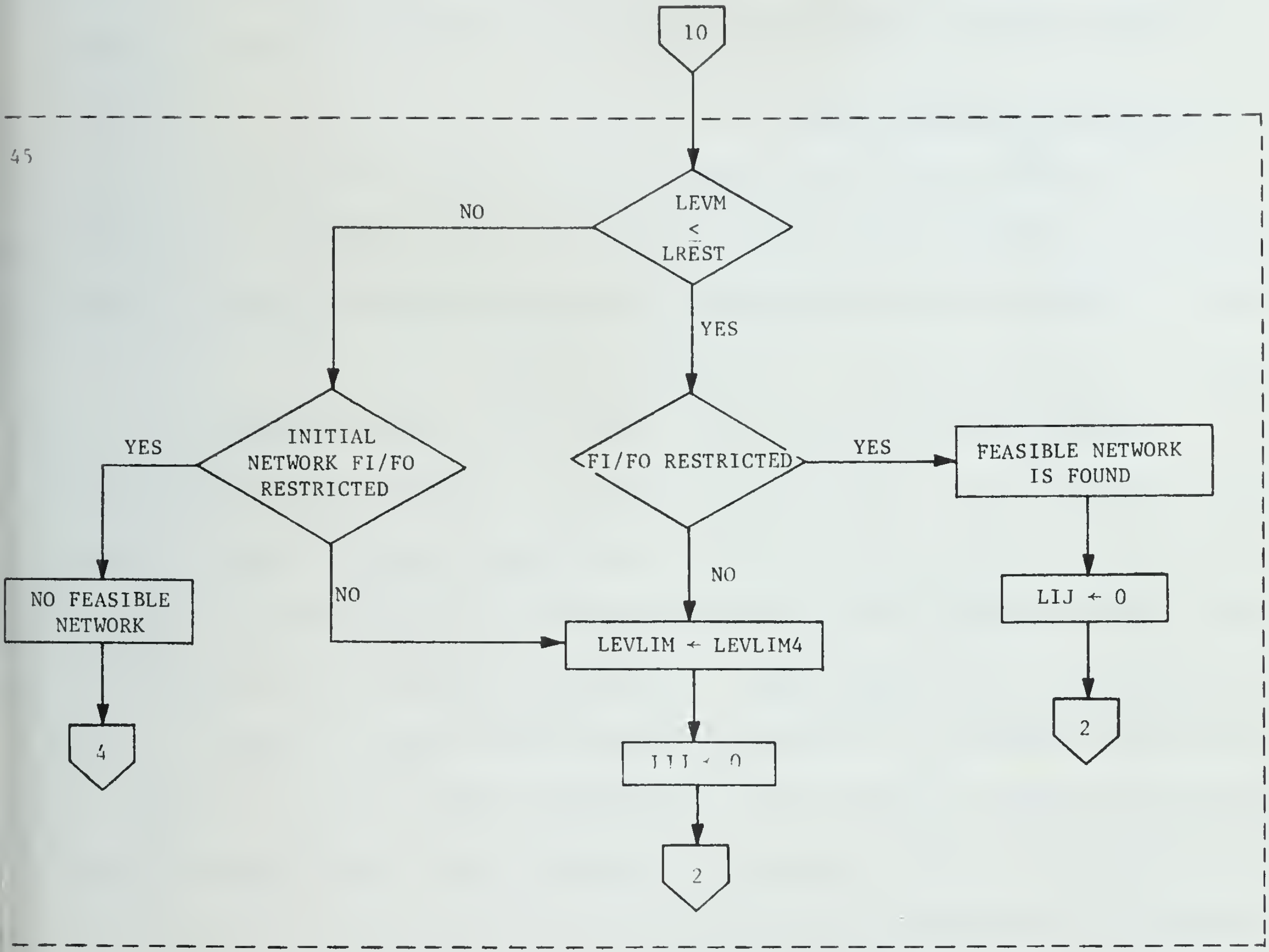
(c) Non-fan-in/non-fan-out restricted transduction step.



(d) Fan-in/fan-out restricted transformation step.



(e) Fan-in/fan-out restricted transduction step.



(f) Control flow for considering the level restriction.

Fig. 4.2-3 Detailed flowchart for the control subroutine MAIN.

current run is finished, so print the summary table and then stop.

Block 44: Check whether GDCST < BSTCST or not. If it is, then go to block 23. Otherwise go to block 6.

Block 45: This block gives the control flow for problems with both fan-in/fan-out restrictions and level-restriction. Details about the control flow is explained already in this section and hence is omitted here.

4.3 Overlay structured program

Overlay structured programming is used when the total memory needed for the program and data exceeds the maximum main storage available. Before explainin what an overlay structured program is and how to design an overlay structured program, the following concepts are introduced first.

Ordinarily, the preparation for executing a source program can be illustrated in Fig. 4.3.1. The input to the language translator is a source module; the output from the language translator is an object module. Before an object module can be executed, it must be processed by the linkage editor. The output of the linkage editor is a load module. The source module can be any program written in symbolic languages like assembly languages, ALGOL, COBOL, FORTRAN, PL/1, APL, etc. The language translator can be an assembler or a compiler. An object module is in relocatable format in unexecutable machine code. A load module is also relocatable, but in executable machine code.

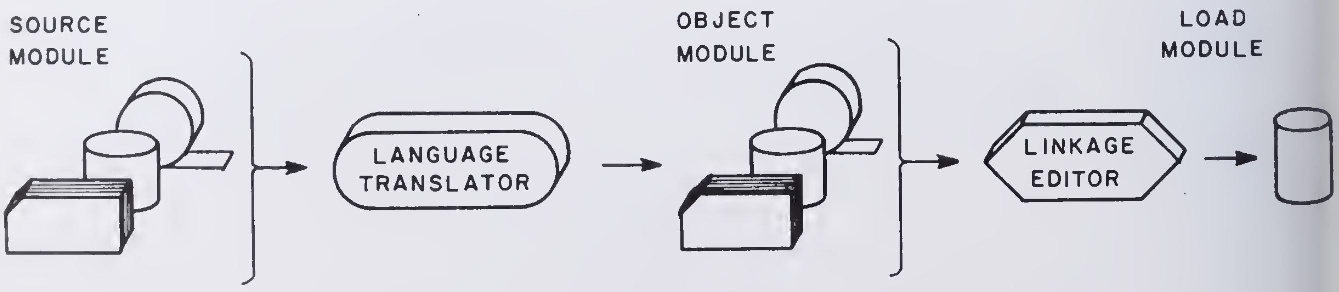


Fig. 4.3.-1 Preparing a load module for execution.

Any module is composed of one or more control sections. A control section is a unit of coding (instructions and data) that is, in itself, an entity. All elements of a control section are loaded and executed in the order specified by the programmer. A control section is, therefore, the smallest separately relocatable unit of a program. For example, a FORTRAN program may contain many subroutines (including the subroutine MAIN), each of which is a control section.

In processing object and load modules, the linkage editor assigns consecutive relative addresses to all control sections and resolves all references between control sections. Object modules produced by several different language translators can be used to form one load module.

Ordinarily, when a load module produced by the linkage editor is executed, all of the control sections of the module remain in main storage throughout execution. The length of the load module is, therefore, the sum of the lengths of all of the control sections. When the main storage space is large enough, this is the most efficient way to execute a program. However, if a program approaches the limit of the main storage available, the programmer should consider using the overlay facilities of the linkage editor before rewriting the program. When the linkage editor overlay facility is requested, the load module is structured so that, at execution time, certain control sections are loaded only when referenced.

The way in which an overlay module is structured depends on the relationships among the control sections within the module, and two control sections which do not have to be in storage at the same time can overlay each other. They can be assigned the same load addresses and are loaded only when referenced.

Control sections are grouped into segments. The control sections required all the time are grouped into a special segment called the root segment. This segment remains in storage throughout execution of an overlay structured program. All other control sections which can be overlayed are grouped into segments separately. When a particular segment is to be executed, any segments between it and the root segment must also be in storage.

For example, assume that a program contains seven control sections (see Fig. 4.3.2), A through G, and exceeds the amount of maximum storage available for its execution. Before the program is rewritten, it is examined to see whether or not it can be placed into an overlay structure. Assume that the relationships among control sections are shown in Fig. 4.3-2(a). In Fig. 4.3-2(a), an arrow from control section i to control section j means that control section j receives control from control section i (or in FORTRAN, subroutine j is called by subroutine i). Since control sections A and B appear in each independent control group, they can be placed in the root segment. Control section C, F and G are in three separate segments, and control sections D and E are in one segment. The overlay structure for this example is shown in Fig. 4.3-2(b). The total length of this overlay structured program is the length of the longest path of the above overlay structure.

The overlay structure shown in Fig. 4.3-2(b) is called the single-region overlay structure. Usually, if a control section appears in several paths in Fig. 4.3-2(b), it is desirable to place that control section in the root segment. However, the root segment can get so large that the benefits of overlay structure are lost. So the multiple-region overlay structure is introduced. For example, Fig. 4.3-3(a) shows a single-region overlay structure. As can be easily seen, control sections H and I are controlled (or called) by control sections F and G. We can place control sections H and I in the root segment. But this will make the root segment longer than necessary (H and I are not used in the path A → B → C → D → E). Fig. 4.3-3(b) shows the multiple-region overlay structure; in it, control sections H and I are overlayed in region 2. Control sections H and I can be placed in the main storage only when control section F or G is in

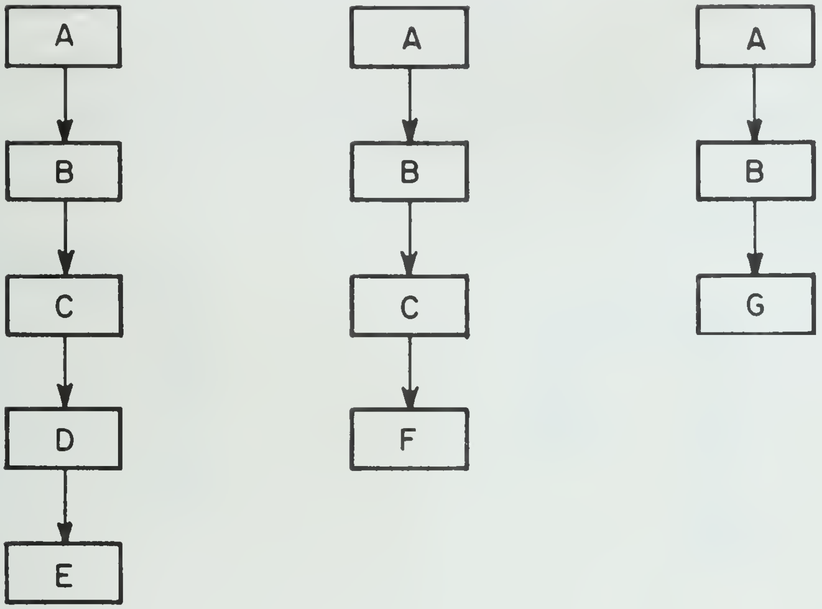


Fig. 4.3-2(a) This program contains seven control sections which are grouped as three independent control section groups

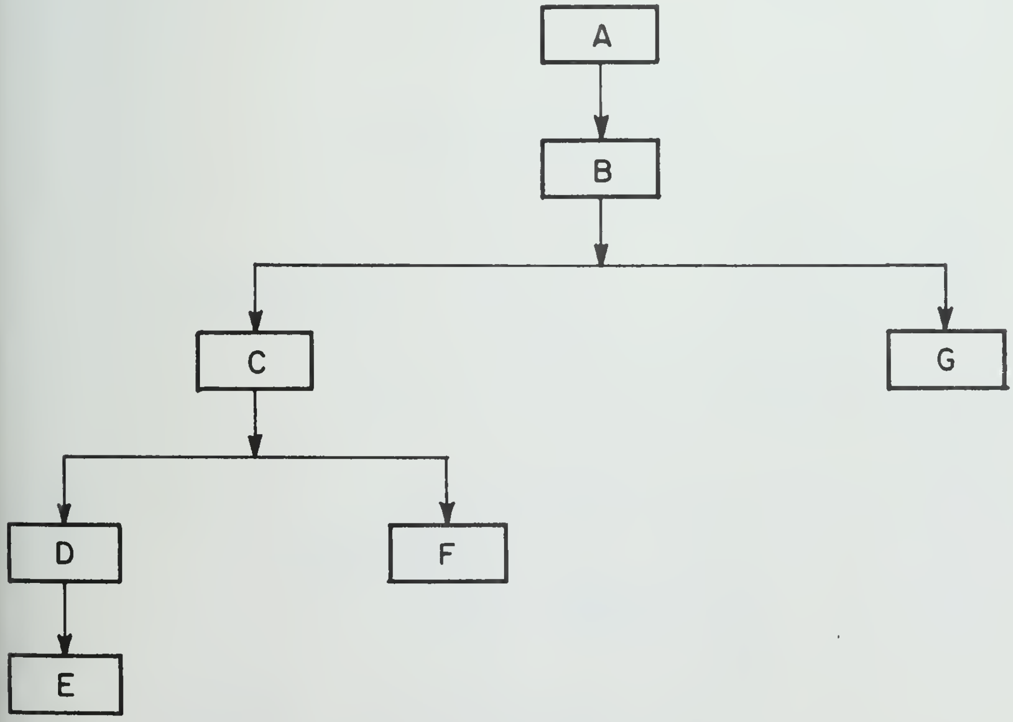


Fig. 4.3-2(b) Single-region overlay structure

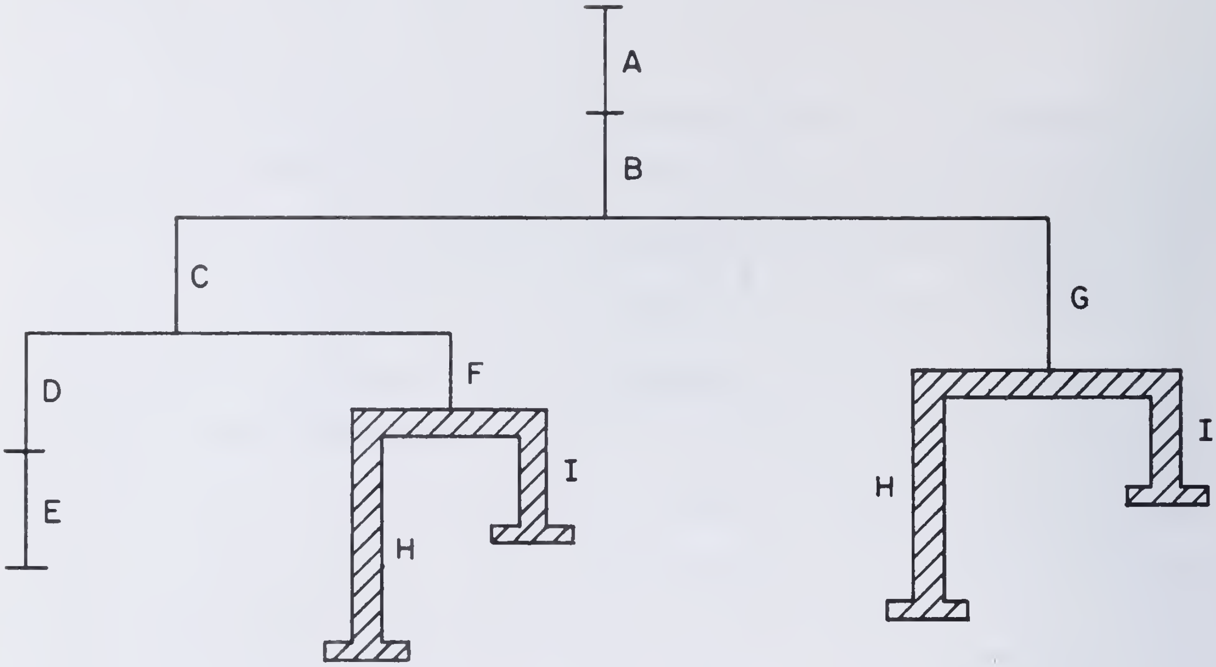


Fig. 4.3-3(a) Control sections H and I appear in more than one path

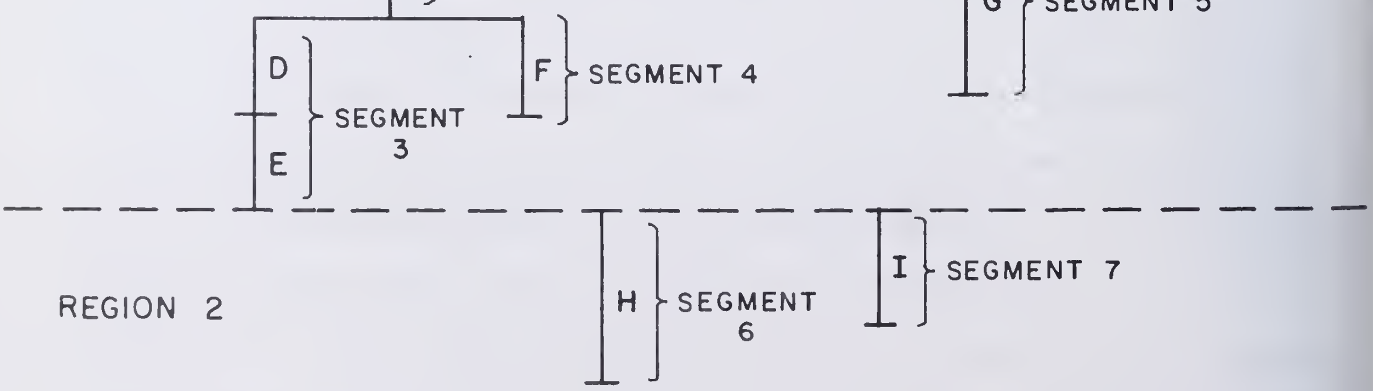


Fig. 4.3-3(b) Multiple-region overlay structure

the main storage.

In the NETTRA system, only single-region overlay structure is used. Table 4.3-1 gives the size of each subroutine. The total length of these subroutines is much greater than the storage available (400 K bytes in decimal normally*). Fig. 4.3-4 shows the single-region overlay structure for the NETTRA system. The length of the longest path in Fig. 4.3-4 is 366 K bytes (in decimal).

*It can be extended to 500 K on special request.

Table 4.3-1 Memory sizes of all subroutines included in the NETTRA system

Type of Subroutines	Memory Size (Bytes)	
	Hexadecimal	Decimal
MAIN	94D8	38,104
COMMON DATA AREA	2EDD8	192,232
I/O SUPPORTING SUBROUTINES	AABC	27,068
BANDB et al	6970	26,992
UNIVSA	57E	1,406
THRLEV	10DA	4,314
TISON et al	10450	66,640
TISLEV et al	15CC	5,580
TANT et al	1AF80	110,432
EXNT	6D6	1,750
JEFF et al	428E	17,028
PROCIP (0)	1854	6,268
PROCIP (I)	E3C	3,644
PDTCNT	CD4	3,284
PROCI	E32	3,634
PRIFF & PROCIV et al	D8D8	55,512
GIMERG	7144	29,252
PROCV et al	A224	41,508
PROCCE et al	DF30	57,138
PARMP & OPTTYP *	1042	4,162

*OPTTYP is an I/O supporting subroutine not called by MAIN very often.

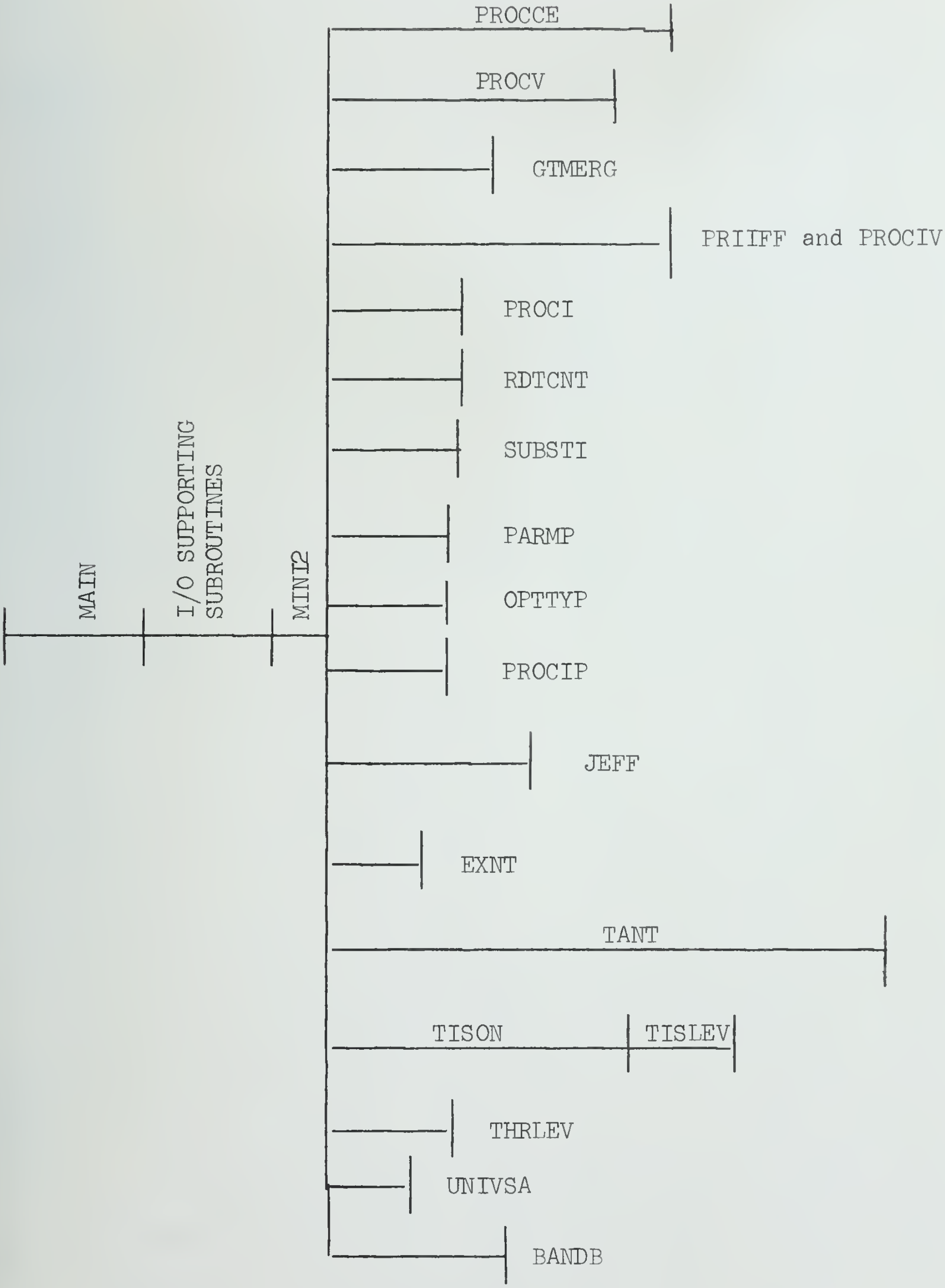


Fig. 4.3-4 Single-region overlay structure for the NETTRA system

5. Statistics and Experimental Results

The IBM 360/75J is used to run experiments for testing the NETTRA system.

It is easy to see from Fig. 4.2-1 that we can design an infinite number of control sequences to solve problems under fan-in/fan-out restrictions. Each control sequence consists of one or more initial network subroutines and a TT-sequence, and each TT-sequence consists of the fan-in/fan-out restricted transformation subroutine and one or more transduction subroutines. For any initial network derived by some initial network subroutine, the TT-sequence and each transduction subroutine in the TT-sequence can be applied as many times as we like.

If we apply the same TT-sequence on different initial networks the same number of times, or if we apply different TT-sequences on the same initia network the same number of times, then we will usually obtain different result Therefore a choice of an appropriate control sequence is an important problem.

Many experiments are made in order to find out the influence of initial network subroutines on the final results and the general tendency of effectiveness and efficiency of the transduction subroutines. The results of experiments are given in this chapter. We can find out how to design appropriate control sequences from these results. Besides, four optional control sequences (OPTION 1 through OPTION 4) are provided for those who are not interested in knowing the details about how to specify a control sequence. OPTION 1 is designed to produce a near-optimal network with very good cost, though the computation time spent by this control sequence may be very long.

TYPE OF INITIAL NETWORK METHODS / COST & TIME / FUNCTION (HEXADECIMAL)	UNIVSA		BANDB		THRLEV		TISON		TANT[†]	
	COST[ξ]	TIME[*]	COST	TIME	COST	TIME	COST	TIME	COST	TIME
1. 4FA295F6	33305	4	19073	87	20102	10	14045	27	12041	5500
2. A6CDDF18	33305	4	20069	82	25100	7	14045	34		
3. FF68A1F3	33303	4	29099	185	25105	9	13040	15		
4. 1EE65240	33310	4	12043	47	17065	7	13035	35		
5. 9E638E7F	33301	5	17058	75	22092	7	11036	14	11036	831
6. 0A888103	33315	4	14043	55	18081	7	13031	34		
7. 49F363CD	33305	4	15052	67	25100	7	14045	45		
8. 8B5809F0	33310	5	15049	57	21086	7	14039	44		
9. BFD6C6DA	33302	5	14059	59	22085	7	14048	14	13041	1531
10. C6E7103E	33307	5	15051	64	26106	7	12037	34	12037	746

[*] Time unit in centiseconds.

[†] For TANT, we cannot get any result in 1 minute for 6 functions.

[ξ] Cost = 1000 × R + C; R = no. of gates, C = no. of connections

OPTION 2 is designed to produce a network with a reasonably good cost in a reasonably short time. OPTION 3 is designed to produce any network in a very short time; in this case, how good the cost of the network is not important. OPTION 4 is a special control sequence designed for producing feasible networks for problems under both fan-in/fan-out and level restrictions.

5.1 Comparison of Initial Network Methods

In order to find out the influence of initial networks on the final results after applying the same TT-sequence the same number of times, ten 5-variable single-output functions are used to do experiments. First, the initial networks obtained by the five initial network methods* are shown in Table 5.1-1. In Table 5.1-1, the given functions are shown in the first column in hexadecimal representation. The cost of a network is defined as 1000 × R + C where R is the number of gates and C is the number of connections in the network. Only uncomplemented external variables are permitted as inputs.

It is observed that subroutine UNIVSA is the least time-consuming one whereas subroutine TANT is the most time-consuming one. Subroutine TANT cannot produce any result within 1 minute execution for most functions. The reason is that subroutine TANT aims at finding "optimal three-level" networks; it becomes very time-consuming when the number of gates in a network exceeds approximately 10. Subroutine TISON usually produces networks with very good costs, although these networks are always three-level networks (or two-level networks if both complemented and uncomplemented external variables are permitted as inputs). Subroutine THRLEV also produces three-level networks, but the

*The results derived by TISLEV will be shown in Section 5.3.

costs of networks derived by THRLEV are usually higher than those by TISON. Networks obtained by surbroutine BANDB are not restricted to be three-level only, and they usually have reasonably good costs.

Table 5.1-1 only gives some ideas how the initial networks obtained by different methods look like and how much computation time different methods spend. It does not show the influence of the initial networks on the final results after applying the control sequences. So some other experiments are made. Since subroutine TANT is too time-consuming, it is not included in later experiments.

For the transduction subroutines which can treat problems under fan-in/fan-out restrictions*, the control sequences corresponding to the flowchart shown in Fig. 5.1-1 are applied. Each control sequence consists of four initial network subroutines and a TT-sequence, and each TT-sequence consists of subroutine JEFF and one of the following 8 transduction subroutines: MINI2, SUBSTI, RDTCNT, PROCI, GTMERG, PRIIFF, PROCIV and PROCCE. Starting from the initial network derived by one of the subroutines UNIVSA, THRLEV, BANDB, and TISON, we will repeatedly apply the selected transduction subroutine under no fan-in/fan-out restrictions until there is no further improvement in the cost. We will then apply subroutine JEFF to transform the network into fan-in/fan-out restricted form. Again we will repeatedly apply the same transduction subroutine under fan-in/fan-out restrictions until there is no further improvement in the network cost. The selected TT-sequence will also be applied repeatedly until there is no further improvement in the cost.

The fan-in/fan-out restrictions for the experiments are set as FI = FO = FOX = FOO = 4. Costs of the resulting networks and computation

*Only subroutine PROCV cannot treat problems under fan-in/fan-out restrictions.

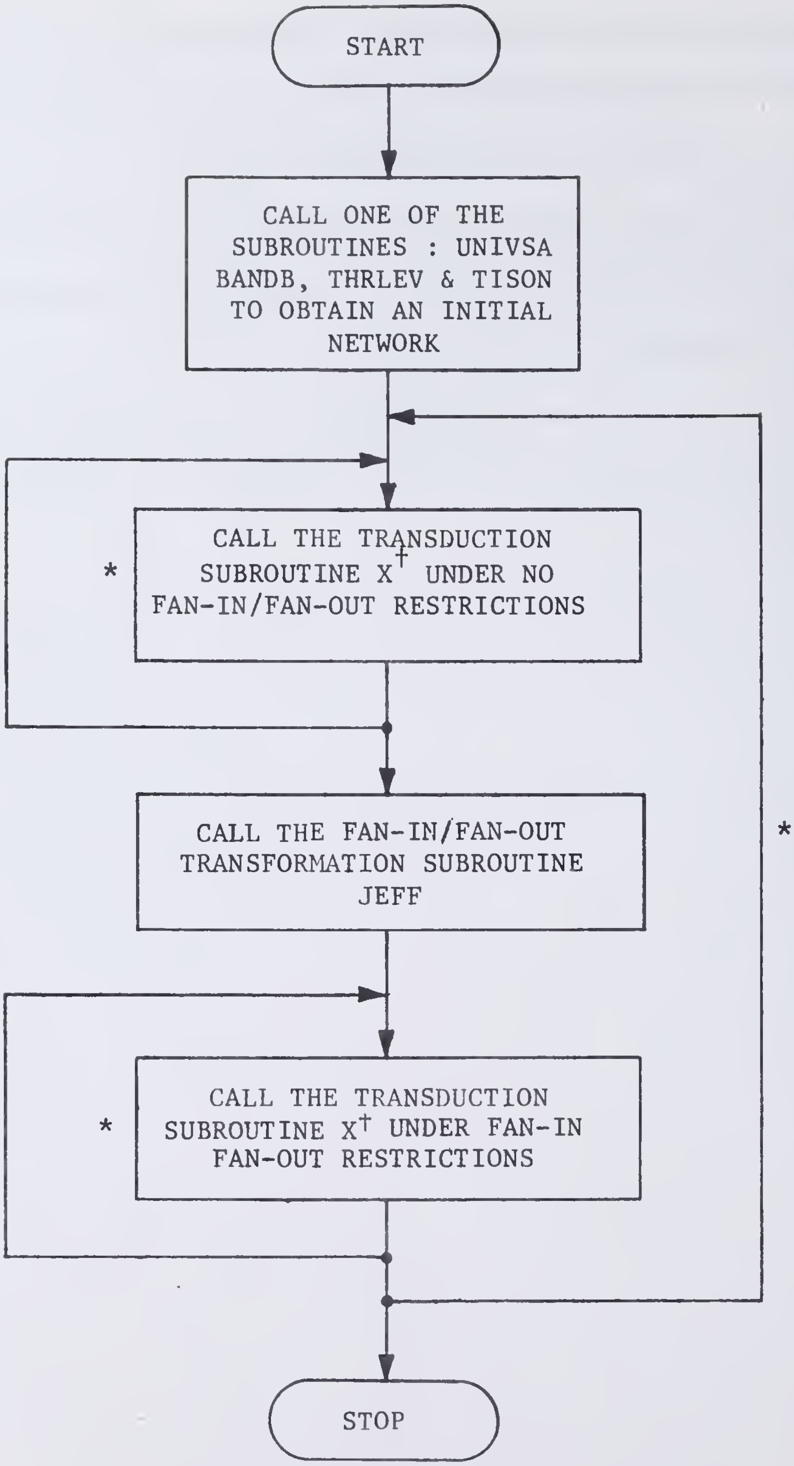

START
CALL ONE OF THE SUBROUTINES : UNIVSA BANDB, THRLEV & TISON TO OBTAIN AN INITIAL NETWORK
CALL THE TRANSDUCTION SUBROUTINE X† UNDER NO FAN-IN/FAN-OUT RESTRICTIONS
*
CALL THE FAN-IN/FAN-OUT TRANSFORMATION SUBROUTINE JEFF
*
CALL THE TRANSDUCTION SUBROUTINE X† UNDER FAN-IN FAN-OUT RESTRICTIONS
*
STOP

Fig. 5.1-1 Flowchart for the experiments for finding out the influence of initial networks.

*These loops will be repeatedly applied until there is no further improvement in the network cost.

times are shown in Table 5.1-2 through Table 5.1-9. Since the same 10 functions as in Table 5.1-1 are used, only the function numbers are shown. The cost of the best network for each function is marked with circles. In Table 5.1-2, the number of best networks obtained by starting from initial networks obtained by UNIVSA, THRLEV, BANDB and TISON are 1, 1, 4 and 5, respectively. This means that the initial networks obtained by BANDB and TISON are "desirable" for the transduction subroutine MINI2, i.e., if MINI2 is used in the experiments following the flowchart shown in Fig. 5.1-1 to simplify the initial network obtained by BANDB or TISON, then we will usually obtain better results. In Table 5.1-3, the networks derived by BANDB and UNIVSA are more desirable for the transduction subroutine SUBSTI. In Table 5.1-4 and Table 5.1-5, the networks obtained by BANDB and TISON are more derivable for the transduction subroutines RDTCNT and PROCI. For transduction subroutines GTMERG, PRIFF and PROCIV, networks obtained by BANDB are more desirable. For the transduction subroutine PROCCE, networks obtained by UNIVSA, BANDB or TISON are more desirable. Table 5.1-10 gives the number of best results obtained by starting from different initial networks and then applying different transduction subroutines. It is easy to see that the initial networks obtained by subroutine TISON are more desirable for the transduction subroutines realizing pruning procedures (including MINI2, RDTCNT and PROCI). For the transduction subroutines which realize the transduction procedures based on gate merging, gate substitution and connectable and disconnectable functions (including SUBSTI, GTMBERG, PRIIFF and PROCIV), networks obtained by BANDB are more desirable. The last row in Table 5.1-10 gives the total number of best networks obtained by starting one of four initial networks, i.e., the sum of the numbers of best methods in each column in Table 5.1-10. It shows that the initial networks obtained by subroutine BANDB are in general more desirable for applying transduction

Table 5.1-2 Results for the experiments shown in Fig. 5.1-1-1. The transduction subroutine MINI2 is used.

COST & TIME	INITIAL NETWORK / FUNCT. NO.	UNIVSA	THRLEV	BANDB	TISON
NETWORK COST	1	23052	29059	$\boxed{21051}$	22051
	2	30063	34065	20071	$\boxed{21050}$
	3	20044	23045	35089	$\boxed{17041}$
	4	18038	18038	18045	$\boxed{15037}$
	5	23050	25049	23059	$\boxed{15038}$
	6	$\boxed{13035}$	$\boxed{13035}$	18047	15033
	7	23052	24054	$\boxed{17046}$	21050
	8	19043	17039	$\boxed{16044}$	20043
	9	22051	33060	$\boxed{17045}$	22054
	10	21044	28057	15037	$\boxed{14039}$
COMPUTATION TIME (CS)*	1	133	114	165	82
	2	172	165	206	90
	3	140	97	358	65
	4	120	69	116	65
	5	125	107	166	64
	6	89	55	105	64
	7	131	108	132	94
	8	112	70	122	95
	9	146	145	132	74
	10	129	135	128	67

* CS = Centiseconds

Table 5.1-3 Results for the experiments shown in Fig. 5.1-1-1. The transduction subroutine SUBSTI is used.

COST TIME	INITIAL NETWORK FUNCT. NO.	UNIVSA	THRLEV	BANDB	TISON
NETWORK COST	1	22051	23053	16044	22051
	2	27060	26057	24064	21050
	3	14039	15037	33089	17041
	4	15034	15034	18045	15037
	5	21048	19043	20057	15038
	6	11033	11033	14041	15033
	7	20050	20050	16044	21050
	8	17041	17039	16044	20043
	9	22051	22048	17045	22054
	10	16040	24052	13037	13029
COMPUTATION TIME (CS)	1	204	203	240	196
	2	297	265	317	189
	3	134	152	518	120
	4	206	154	142	101
	5	227	203	167	103
	6	120	71	159	88
	7	185	179	147	188
	8	202	103	128	163
	9	255	236	140	188
	10	184	199	154	100

* CS = Centiseconds

Table 5.1-4 Results for the experiments shown in Fig. 5.1-1-1. The transduction subroutine RDTCNT is used.

COST & TIME	INITIAL NETWORK FUNCT. NO.	UNIVSA	THRLEV	BANDB	TISON
NETWORK COST	1	29057	28056	20052	22051
	2	35065	34064	26065	21050
	3	22044	21043	23051	17041
	4	18038	18038	18045	15037
	5	24050	25049	19048	15038
	6	12033	13037	17043	15033
	7	24053	25053	17042	21050
	8	18038	15037	14040	20043
	9	29056	33060	17045	22054
	10	26049	29057	15039	14039
COMPUTATION TIME (CS)*	1	305	175	245	101
	2	340	216	324	95
	3	211	136	491	84
	4	255	81	137	85
	5	205	146	248	65
	6	170	80	161	65
	7	227	161	182	113
	8	238	98	165	114
	9	302	195	165	98
	10	277	183	147	74

* CS = Centiseconds

Table 5.1-5 Results for the experiments shown in Fig. 5.1-1-1. The transduction subroutine PROCI is used.

T TIME	FUNCT. NO.	INITIAL NETWORK UNIVSA	THRLEV	BANDB	TISON
NETWORK COST	1	23052	27054	20052	22051
	2	30061	34065	26065	21050
	3	20044	21043	23051	17041
	4	16037	18038	18045	15037
	5	23050	25049	19048	15038
	6	15037	13033	17043	15033
	7	26057	25053	17042	21050
	8	14035	15036	14040	20043
	9	23052	33060	17045	22054
	10	23050	29057	15039	14039
COMPUTATION TIME (CS)*	1	743	185	224	87
	2	720	217	322	99
	3	382	162	486	76
	4	618	77	127	70
	5	264	144	226	60
	6	470	71	156	69
	7	458	149	176	117
	8	659	83	154	115
	9	690	209	157	83
	10	664	205	151	84

* CS = Centiseconds

Table 5.1-6 Results for the experiments shown in Fig. 5.1-1-1. The transduction subroutine GIMERG is used.

COST & TIME	INITIAL NETWORK FUNCT. NO.	UNIVSA	THRLEV	BANDB	TISON
NETWORK COST	1	20049	22053	19049	22051
	2	28063	23057	19050	
	3	15038	15038	23062	17041
	4	15034	15034	18045	15037
	5	21048	19046	20057	15038
	6	11033	11033	10029	15033
	7	22053	18045	16044	21051
	8	14035	15036	16044	14037
	9	22051	23052	17045	22054
	10	16040	23052	13037	14039
COMPUTATION TIME (CS)*	1	415	499	361	314
	2	759	643	545	260
	3	255	228	664	180
	4	324	242	208	152
	5	364	382	254	144
	6	155	106	186	153
	7	405	352	151	309
	8	334	186	154	177
	9	500	538	209	318
	10	280	556	191	143

* CS = Centiseconds

Table 5.1-7 Results for the experiments shown in Fig. 5.1-1-1.
The transduction subroutine PRIIFF is used.

T TIME	INITIAL NETWORK FUNCT. NO.	UNIVSA	THRLEV	BANDB	TISON
NETWORK COST	1	19047	20049	14037	17041
	2	19045	17040	20050	21050
	3	13034	13034	13036	17041
	4	13033	17037	13031	15037
	5	13036	15040	14037	15038
	6	11033	11033	10026	12027
	7	17043	15040	14039	17039
	8	16039	13033	11030	14037
	9	19045	19045	16035	20047
	10	16039	16041	13032	14039
COMPUTATION TIME (CS)*	1	1186	469	545	458
	2	1882	593	488	330
	3	872	389	1009	445
	4	1383	322	362	150
	5	846	628	487	165
	6	769	207	243	153
	7	1026	490	367	515
	8	1091	325	330	240
	9	1448	421	602	639
	10	1251	567	309	138

* CS = Centiseconds

Table 5.1-8 Results for the experiments shown in Fig. 5.1-1-1. The transduction subroutine PROCIV is used.

COST & TIME	FUNCT. NO. / INITIAL NETWORK	UNIVSA	THRLEV	BANDB	TISON
NETWORK COST	1	17044	18044	14039	17041
	2	17042	17039	17045	18041
	3	13034	13034	15034	17044
	4	15035	14036	17037	15037
	5	13032	14037	14036	15038
	6	11033	11033	11031	15033
	7	16044	16044	11033	13034
	8	11030	12030	11029	14030
	9	15040	18044	17043	17044
	10	16039	14039	13034	14039
COMPUTATION TIME (CS)*	1	3139	2287	2323	2162
	2	3296	2900	2842	3708
	3	1674	1391	2677	1179
	4	2789	3019	3573	849
	5	3662	1775	2220	891
	6	1091	809	1082	781
	7	1969	1762	1466	3205
	8	1534	1236	1385	2143
	9	3115	2174	2426	2870
	10	3711	1518	1177	772

* CS = Centiseconds

Table 5.1-9 Results for the experiments shown in Fig. 5.1-1-1. The transduction subroutine PROCCE is used.

OST & TIME	INITIAL NETWORK FUNCT. NO.	UNIVSA	THRLEV	BANDB	TISON
NETWORK COST	1	17043	20051	16043	18045
	2	17046	16044	16047	16043
	3	11033	12037	14035	13035
	4	12034	12034	13038	15037
	5	15034	15038	13039	13032
	6	10031	10031	9027	12029
	7	16044	13039	13036	16044
	8	12029	13032	12030	13032
	9	15041	17044	17045	14034
	10	14037	15041	14039	14039
COMPUTATION TIME (CS)*	1	2663	3795	2046	6690
	2	3410	2563	3067	8134
	3	1273	1479	2860	3513
	4	1150	853	935	1697
	5	4778	1921	2973	2704
	6	781	573	1039	1195
	7	2196	1855	1525	4027
	8	2437	2424	1406	1477
	9	4570	4154	2407	6288
	10	1508	3494	1749	1233

* CS = Centiseconds

Table 5.1-10 Total number of best methods obtained by starting from one of four initial networks and applying one of eight transduction subroutines.

Transduction Subroutine Applied	Initial Network Subroutine Applied			
	UNIVSA	THRLEV	BANDB	TISON
MINI2	1	1	4	5
SUBSTI	3	2	5	2
RDTCNT	1	0	4	5
PROCI	1	1	3	5
GTMERG	3	2	6	1
PRIIFF	2	2	7	0
PROCIV	3	3	5	0
PROCCE	4	1	3	3
Total Number of Best networks	18	11	37	21

procedures than the initial networks obtained by any other initial network subroutine.

Table 5.1-11 Average computation time for finding initial networks by different initial network subroutines

UNIVSA	THRLEV	BANDB	TISON
4.4 cs	7.5 cs	77.8 cs	29.6 cs

The average computation time for finding initial networks is shown in Table 5.1-11. It is obtained by adding up the computation time in each column in Table 5.1-1 and then dividing the sum by 10. The average computation time for the experiments shown in Fig. 5.1-1 is given in the upper half of Table 5.1-12, it is obtained by adding up the computation time in each column in each table of Table 5.1-2 through Table 5.1-9 and then dividing the sum by 10. The percentage of average computation time for finding initial networks is given in the lower half of Table 5.1-12. These results indicate that the transduction subroutines which realize the transduction procedures based on connectable and disconnectable functions and error-compensation are usually more time-consuming than other transduction subroutines; in other words, the computation time for finding initial networks is only a very small fraction of the total computation time when these transduction subroutines are applied in the experiments.

From the previous experimental results and analyses, we get the following conclusions:

(1) The costs of the initial networks do not have direct relationship with the final results; i.e., starting from initial networks with lower costs do not imply that the corresponding networks also have lower

Table 5.1-12 Average computation time for the experiments in Fig. 5.1-1

TRANSDUCTION SUBROUTINE USED	AVERAGE COMPUTATION TIME / 10 FUNCTIONS (CS)			
	UNIVSA	THRLEV	BANDB	TISON
MINI2	129.7	106.5	163.0	76.0
SUBSTI	201.4	176.5	211.2	143.6
RDTCNT	232.5	147.1	225.4	89.4
PROCI	500.4	150.2	233.6	86.0
GTMERG	379.1	373.2	292.3	215.0
PRIIFF	1175.5	441.1	474.2	323.3
PROCIV	2598.0	1887.1	2115.1	1856.0
PROCCE	2476.6	2311.1	2000.7	3695.8

TRANSDUCTION SUBROUTINE USED	PERCENTAGE OF COMPUTATION TIME FOR FINDING INITIAL AVERAGE NETWORK (%)			
	UNIVSA	THRLEV	BANDB	TISON
MINI2	3.39	7.04	47.73	38.94
SUBSTI	2.18	4.25	36.84	20.61
RDTCNT	1.89	5.09	34.51	33.10
PROCI	0.88	4.99	33.30	34.42
GTMERG	1.16	2.01	26.62	13.76
PRIIFF	0.37	1.70	16.41	9.16
PROCIV	0.17	0.40	3.68	1.59
PROCCE	0.18	0.32	3.89	0.80

costs after applying the transduction procedures. Sometimes the final results are more influenced by the network configurations rather than the costs of the initial networks. More specifically speaking, Table 5.1-1 shows that the initial networks obtained by Tison's method usually have lower costs than those obtained by other methods. But Table 5.1-10 shows that starting from the initial networks obtained by the branch-and-bound method, better final results can usually be obtained. This is because the initial networks obtained by the branch-and-bound method are usually multiple-level networks, and hence are more suitable for the transduction procedures to reconfigure. The initial networks obtained by Tison's method are restricted to be two-level or three-level, and hence are more difficult to reconfigure, even though they have lower costs. The initial networks obtained by three-level network method are also restricted to be two-level or three-level; since they usually have higher costs than the initial networks obtained by Tison's method, the corresponding final results are worse. The initial networks obtained by the universal network method are multiple-level, but they usually contain too many gates and connections and hence are more difficult to simplify.

(2) The computation time spent in finding initial networks is much shorter than that in applying the transduction and transformation subroutines. This is especially true when those sophisticated transduction subroutines, such as PRIIFF, PROCIV and PROCCE, are applied.

5.2 Comparison of Transduction Procedures

The results obtained in Table 5.1-2 through Table 5.1-9 can also be used for analyzing the effectiveness and the efficiency of the transduction procedures. Table 5.2-1 through Table 5.2-4 are rearrangements of the costs of Table 5.1-2 through Table 5.1-9, which are more convenient for us to compare the transduction procedures. In each table, one of four initial network subroutines is used to obtain networks, and then different transduction subroutines are applied (according to the flowchart in Fig. 5.1-1) to simplify the initial networks. Best results for each function are marked with circles.

In Table 5.2-1, if subroutine PROCIV or subroutine PROCCE is applied, then we can obtain 5 or 6 best networks, respectively. In Table 5.2-2, corresponding to subroutines PROCIV and PROCCE, we can get 4 and 6 best results, respectively. In Table 5.2-3, we can obtain 5, 2 and 3 best results corresponding to subroutines PRIIFF, PROCIV and PROCCE, respectively. In Table 5.2-4 the situation is very interesting. For Function 4 and Function 10, we can get best results no matter what transduction subroutines are involved. But in general, subroutine PROCCE is still more effective. Since the initial networks obtained by subroutine TISON are based on the minimum sums, they usually do not have redundant connections*. Only those complex transduction procedures, like PRIIFF, PROCIV and PROCCE, can significantly reconfigure and simplify the networks. Table 5.2-5 gives the number of best results that we can get if certain transduction procedures are applied. The last row in Table 5.2-5 gives the total number of best results obtained by each transduction subroutines (i.e., it is the sum of numbers in each column). This shows that in general subroutines PROCCE, PROCIV and PRIIFF are more desirable to use in deriving near-optimal solutions.

Another interesting statistic is the "average cost" obtained corresponding to each transduction subroutine. It is derived by adding up the numbers of gates and the numbers of connections separately in each column of each

*This is also true even after applying the fan-in/fan-out restricted transformations.

for the experiments shown in Fig. 5.1-1, initial network subroutine UNIVSA is applied.

FUNCTION NUMBER \ TRANSDUCTION SUBROUTINE APPLIED	MINI2	SUBSTI	RDTCNT	PROCI	GTMERG	PRIIFF	PROCIV	PROCCE
1	23052	22051	29057	23052	20049	19047	17044	[✓ 17043]
2	30063	27060	35065	30061	28063	19045	[✓ 17042]	17046
3	20044	14039	22044	20044	15038	13034	13034	[✓ 11033]
4	18038	15034	18038	16037	15034	13033	15035	[✓ 12034]
5	23050	21048	24050	23050	21048	13036	[✓ 13032]	15034
6	13035	11033	12033	15037	11033	11033	11033	[✓ 10031]
7	23052	20050	24053	26057	22053	17043	[✓ 16044]	[✓ 16044]
8	19043	17041	18038	14035	14035	16039	[✓ 11030]	12029
9	22051	22051	29056	23052	22051	19045	[✓ 15040]	15041
10	21044	16040	26049	23050	16040	16039	16039	[✓ 14037]

Best results for each function are marked with circles.

Table 5.2-2 Network costs for the experiments shown in Fig. 5.1-1, initial network subroutine THRLEV is applied.

TRANSDUCTION SUBROUTINE APPLIED / FUNCTION NUMBER	MINI2	SUBSTI	RDTCNT	PROCI	GTMERG	PRIIFF	PROCIV	PROCCE
1	29059	23053	28056	27054	22053	20049	[✓ 18044]	20051
2	34065	26057	34064	34065	23057	17040	17039	[✓ 16044]
3	23045	15037	21043	21043	15038	13034	13034	[✓ 12037]
4	18038	15034	18038	18038	15034	17037	14036	[✓ 12034]
5	25049	19043	25049	25049	19046	15040	[✓ 14037]	15038
6	13035	11033	13037	13033	11033	11033	11033	[✓ 10031]
7	24054	20050	25053	25053	18045	15040	16044	[✓ 13039]
8	17039	17039	15036	15036	15036	13033	[✓ 12030]	13032
9	33060	22048	33060	33060	23052	19045	18044	[✓ 17044]
10	28057	24052	29057	29057	23052	16041	[✓ 14039]	15041

Best results for each function are marked with circles.

Table 5.2-5 Network costs for the experiments shown in Fig. 5.1-1, initial network subroutine BANDB is applied.

TRANSDUCTION SUBROUTINE APPLIED / FUNCTION NUMBER	MINI2	SUBSTI	RDTCNT	PROCI	GTMERG	PRIIFF	PROCIV	PROCCE
1	21051	16044	28056	20052	19049	14037	14039	16043
2	30071	24064	34064	26065	19050	20050	17045	16047
3	35089	33089	21043	23051	23062	13036	15034	14035
4	18045	18045	18038	18045	18045	13031	17037	13038
5	23059	20057	25049	19048	20057	14037	14036	13039
6	18047	14041	13037	17043	10029	10026	11031	9027
7	17046	16044	25053	17042	16044	14039	11033	13036
8	16044	16044	15036	14040	16044	11030	11029	12030
9	17045	17045	33060	17045	17045	16035	17043	17045
10	15037	13037	29057	15039	13037	13032	13034	14039

Best results for each function are marked with circles.

Table 5.2-4 Network costs for the experiments shown in Fig. 5.1-1, initial network subroutine TISON is applied.

FUNCTION NUMBER \ TRANSDUCTION SUBROUTINE APPLIED	MINI2	SUBSTI	RDTCNT	PROCI	GTMERG	PRIIFF	PROCIV	PROCCE
1	22051	22051	22051	22051	22051	**17041**	**17041**	18045
2	21050	21050	21050	21050	21050	21050	18041	**16043**
3	17041	17041	17041	17041	17041	17041	17044	**13035**
4	**15037**	**15037**	**15037**	**15037**	**15037**	**15037**	**15037**	**15037**
5	15038	15038	15038	15038	15038	15038	15038	**13032**
6	15033	15033	15033	15033	15033	**12027**	15033	12029
7	21050	21050	21050	21050	21051	17039	**13034**	16044
8	20043	20043	20043	20043	14037	14037	14030	**13032**
9	22054	22054	22054	22054	22054	20047	17044	**14034**
10	**14039**	**14039**	**14039**	**14039**	**14039**	**14039**	**14039**	**14039**

Best results for each function are marked with circles.

Table 5.2-5 Number of best networks obtained by different combinations of initial network subroutines and transduction subroutines

TRANSDUCTION SUBROUTINE APPLIED / INITIAL NETWORK SUBROUTINE	MINI2	SUBSTI	RDTCNT	PROCI	GTMERG	PRIIFF	PROCIV	PROCCE
UNIVSA	0	0	0	0	0	0	5	6
THRLEV	0	0	0	0	0	0	4	6
BANDB	0	0	0	0	0	5	2	3
TISON	2	2	2	2	2	4	4	7
TOTAL NUMBER OF BEST RESULTS	2	2	2	2	2	9	15	22

table in Table 5.2-1 through Table 5.2-4 and then dividing the sums by 10. These results are shown in Table 5.2-6. From this table and the average computation time shown in the upper-half of Table 5.1-12, the general tendency of effectiveness and efficiency of each transduction subroutine (or procedure) can be determined. This is shown in Fig. 5.2-1.

Notice that so far we have not compared subroutine PROCV with other transduction subroutines. The reason is that subroutine PROCV was originally designed under no fan-in/fan-out restrictions. Since its performance is not very impressive, it is not modified for treating problems under fan-in/fan-out restrictions. The experiments shown in Fig. 5.2-2 are made to compare subroutine PROCV with subroutines RDTCNT, PROCI and SUBSTI. Table 5.2-7 through Table 5.2-10 give the experimental results. The fan-in/fan-out restrictions are specified as FI = FO = FOX = FOO = 4, and the previous ten 5-variable functions are used. In Table 5.2-7, it is observed that the results obtained by using subroutine PROCV in the experiments are much worse than those by other three transduction procedures; also the computation time is much longer. This is because that subroutine PROCV realizes the transduction procedures based on gate substitution, and each gate (except the output gates) in a universal network realizes only one minterm and therefore no gate can substitute for any other gates. For Functions 2, 4 and 10, the results are not fan-in/fan-out restricted; because the maximum number of gates allowed in a network in the NETTRA system is 60-n, where n is the number of external variables, and for these problems more than 60-n gates are needed to transform the network into fan-in/fan-out restricted form.

For initial networks derived by other three initial network subroutines,

Table 5.2-6 Average costs[*] obtained by each combination of initial network subroutines and transduction subroutines

TRANSDUCTION SUBROUTINE APPLIED / INITIAL NETWORK SUBROUTINE APPLIED	MINI2	SUBSTI	RDTCNT	PROCI	GTMERG	PRIIFF	PROCIV	PROCCE
UNIVSA	21.2 47.2	18.5 44.7	23.7 48.3	21.3 47.5	18.4 44.4	15.6 39.4	14.4 37.3	13.9 37.2
THRLEV	24.4 50.1	19.2 44.6	24.1 49.3	24.0 48.8	18.4 44.6	15.6 39.2	14.7 38.0	14.3 39.1
BANDB	21.0 53.4	18.7 51.0	24.1 49.3	18.6 47.0	17.1 46.2	13.8 35.3	14.0 36.1	13.7 37.9
TISON	18.2 43.6	18.2 43.6	18.2 43.6	18.2 43.6	18.2 43.6	17.6 43.3	15.5 38.1	14.4 37.0

*The average number of gates and the average number of connections are shown in upper and lower rows, respectively, in each cell.

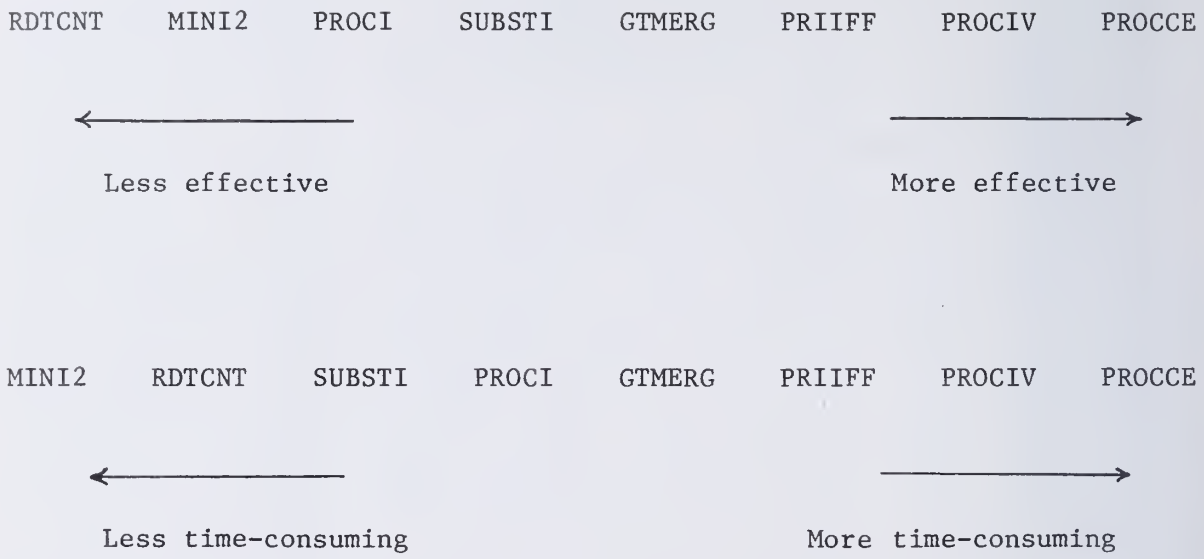

RDTCNT MINI2 PROCI SUBSTI GTMERG PRIIFF PROCIV PROCCE
Less effective
More effective
MINI2 RDTCNT SUBSTI PROCI GTMERG PRIIFF PROCIV PROCCE
Less time-consuming
More time-consuming

Fig. 5.2-1 General tendency of effectiveness and efficiency of the transduction subroutines which can treat fan-in/fan-out restrictions.

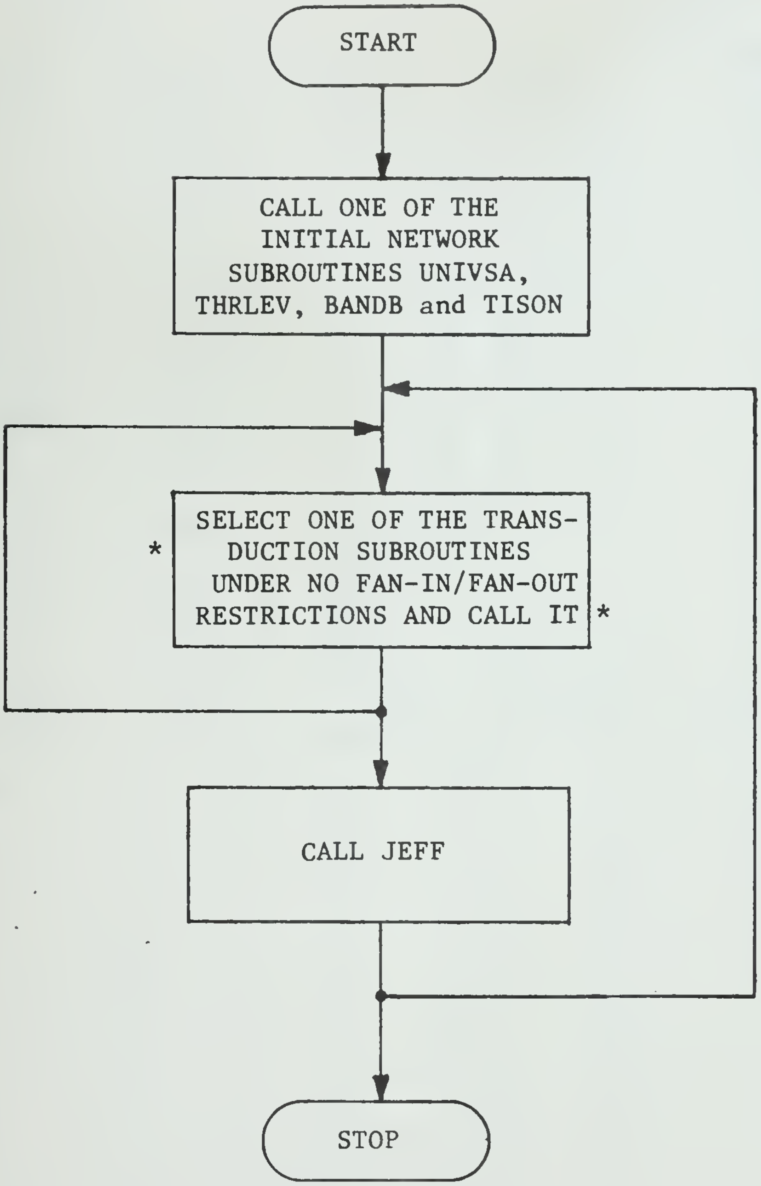

START
CALL ONE OF THE INITIAL NETWORK SUBROUTINES UNIVSA, THRLEV, BANDB and TISON
SELECT ONE OF THE TRANS-DUCTION SUBROUTINES UNDER NO FAN-IN/FAN-OUT RESTRICTIONS AND CALL IT
*
*
CALL JEFF
STOP

Fig. 5.2-2 Experiments for comparing subroutine PROCV with subroutines RDTCNT, PROCI and SUBSTI.

*These loops will be repeatedly applied until there is no further improvement in the cost.

Table 5.2-7 Results for experiments shown in Fig. 5.2-2. Initial network subroutine UNIVSA is used.

	TRANSDUCTION SUBROUTINE APPLIED / FUNC. NO.	RDTCNT	PROCI	SUBSTI	PROCV
COST	1	29057	23052	23052	41087
	2	35065	30061	29062	55138 *
	3	22044	20044	14039	37083
	4	18038	16037	16035	54164 *
	5	24050	23050	22049	34073
	6	12033	15037	11033	21054
	7	24053	26057	20050	36078
	8	18038	14035	18042	53115
	9	29056	23052	23052	54118
	10	26049	23050	16040	55141 *
TIME (CS)	1	289	659	262	3888
	2	313	688	357	2691
	3	190	360	170	2550
	4	215	616	260	3343
	5	167	223	270	3194
	6	176	474	129	2901
	7	194	439	260	3640
	8	248	638	185	4715
	9	255	592	262	2424
	10	245	613	204	2264

* These results are not fan-in/fan-out restricted.

Table 5.2-8 Results for experiments shown in Fig. 5.2-2. Initial network subroutine THRLEV is used.

TRANSDUCTION SUBROUTINE APPLIED	FUNC. NO.	RDTCNT	PROCI	SUBSTI	PROCV
COST	1	28056	27054	23053	26056
	2	34064	34065	27058	29063
	3	21043	21043	15037	19046
	4	18038	18038	16035	20045
	5	25049	25049	21045	20049
	6	13033	13033	11033	13036
	7	25053	25053	20050	28064
	8	15036	15036	17039	16040
	9	33060	33060	23049	23051
	10	29057	29057	24052	17045
TIME (CS)	1	133	162	275	400
	2	172	183	500	446
	3	98	139	164	367
	4	60	64	194	215
	5	114	108	197	312
	6	72	68	77	186
	7	134	131	216	476
	8	79	68	111	288
	9	145	145	277	393
	10	158	168	239	290

* These results are not fan-in/fan-out restricted.

Table 5.2-9 Results for experiments shown in Fig. 5.2-2.

Initial network subroutine BANDB is used.

TRANSDUCTION SUBROUTINE APPLIED	FUNC. NO.	RDTCNT	PROCI	SUBSTI	PROCV
COST	1	20052	20052	16044	26069
	2	26065	26065	26063	25066
	3	23051	23051	34089	23058
	4	18045	18045	18045	15041
	5	19048	19048	20057	26063
	6	17043	17043	15041	11035
	7	17042	17042	16044	21058
	8	14040	14040	16044	16045
	9	17045	17045	17045	21057
	10	15039	15039	13037	21055
TIME (CS)	1	207	202	234	640
	2	273	239	386	1053
	3	455	475	915	1034
	4	100	96	161	157
	5	212	210	198	426
	6	145	128	143	212
	7	137	144	153	340
	8	146	137	130	282
	9	146	125	169	289
	10	131	132	133	305

* These results are not fan-in/fan-out restricted.

transduction subroutine PROCV performs comparably to other three transduction subroutines (i.e., sometimes worse, sometimes better), but it is always more time-consuming.

5.3 Determination for Control Sequences

After finding out the general tendency of effectiveness and efficiency of the transduction subroutines, six TT-sequences are designed and listed in the following table. Some of these TT-sequences consist of more time-consuming but effective transduction subroutines, and some of them consist of less time-consuming transduction subroutines. These TT-sequences are combined with four initial network subroutines to form control sequences. The flowchart shown in Fig. 5.3-1 is followed to do experiments for testing these control sequences. Notice that in Table 5.3-1, each transduction subroutine is followed by a "∞" sign. This means that the selected transduction subroutine will be repeatedly applied until there is no further improvement in the network cost, i.e., the same meaning as the asterisk in Fig. 5.3-1. Besides, the selected TT-sequence is applied only once in the experiments.

Table 5.3-1 Six TT-Sequences

TT-SEQUENCE	TRANSDUCTION SUBROUTINES AND NUMBER OF EXECUTION TIMES (NO FAN-IN/FAN-OUT RESTRICTIONS STEP)	TRANSFORMATION SUBROUTINE	TRANSDUCTION SUBROUTINES (FAN-IN/FAN-OUT RESTRICTED)
TT1	MINI2 ∞ PRIIFF ∞	JEFF	PRIIFF ∞
TT2	MINI2 ∞ PRIIFF ∞	JEFF	PROCIV ∞
TT3	GTMERG ∞ PRIIFF ∞	JEFF	PROCCE ∞
TT4	GTMERG ∞ PRIIFF ∞	JEFF	PROCIV ∞ PROCCE ∞
TT5	MINI2 ∞ GTMERG ∞	JEFF	PROCV ∞ PROCCE ∞
TT6	MINI2 ∞ SUBSTI ∞ GTMERG ∞	JEFF	PRIIFF ∞ PROCIV ∞ PROCCE ∞

Table 5.2-10 Results for experiments shown in Fig. 5.2-2. Initial network subroutine TISON is used.

	TRANSDUCTION SUBROUTINE APPLIED / FUNC. NO.	RDTCNT	PROCI	SUBSTI	PROCV
COST	1	22051	22051	22051	21052
	2	21050	21050	21050	21050
	3	17041	17041	17041	20048
	4	15037	15037	15037	15037
	5	15038	15038	15038	15038
	6	15033	15033	15033	12028
	7	21050	21050	21050	21051
	8	20043	20043	20043	14037
	9	22054	22054	22054	22054
	10	14039	14039	14039	14039
TIME (CS)	1	70	75	159	546
	2	76	73	153	249
	3	49	52	119	262
	4	66	63	96	101
	5	53	50	101	122
	6	60	57	88	101
	7	94	87	170	306
	8	83	80	152	177
	9	77	65	172	257
	10	64	47	80	90

* These results are not fan-in/fan-out restricted.

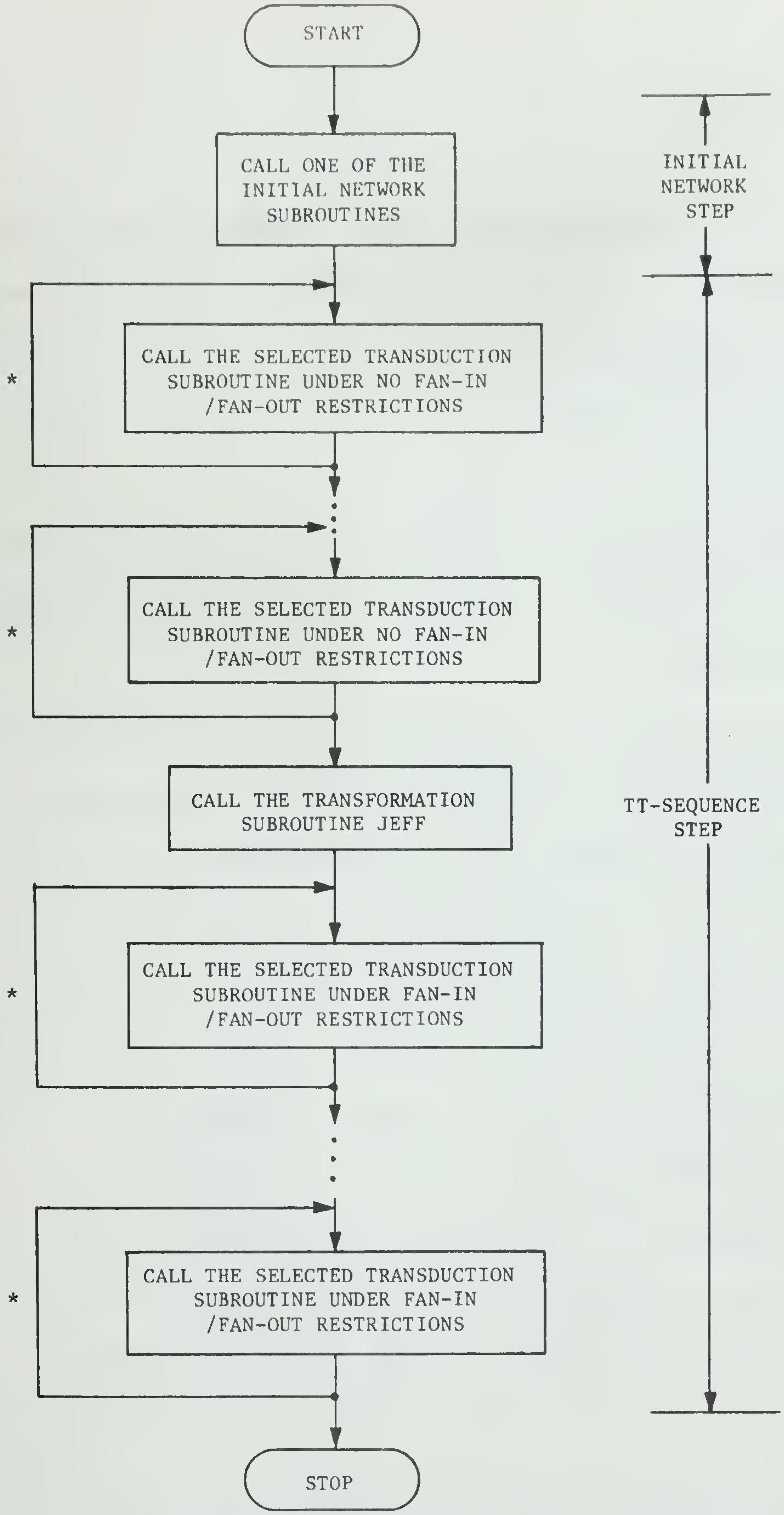

START
CALL ONE OF THE INITIAL NETWORK SUBROUTINES
INITIAL NETWORK STEP
*
CALL THE SELECTED TRANSDUCTION SUBROUTINE UNDER NO FAN-IN /FAN-OUT RESTRICTIONS
*
CALL THE SELECTED TRANSDUCTION SUBROUTINE UNDER NO FAN-IN /FAN-OUT RESTRICTIONS
CALL THE TRANSFORMATION SUBROUTINE JEFF
TT-SEQUENCE STEP
*
CALL THE SELECTED TRANSDUCTION SUBROUTINE UNDER FAN-IN /FAN-OUT RESTRICTIONS
*
CALL THE SELECTED TRANSDUCTION SUBROUTINE UNDER FAN-IN /FAN-OUT RESTRICTIONS
STOP

Fig. 5.3-1 Experiments for testing the TT-sequences shown in Table 5.3-1.

[*] These loops will be repeatedly executed until there is no furter improvement in the network cost.

Thirty 4-variable single-output functions and ten 5-variable single-output functions are used for the experiments. The fan-in/fan-out restrictions, FI, FO, FOX and FOO, are set to the values 3 and 4 for the four-variable and 5-variable functions, respectively. Only uncomplemented external variables are permitted as inputs.

Table 5.3-2 through Table 5.3-5 give the experimental results for thirty 4-variable functions. The best results obtained for each function are marked with circles.

Now let us make the cost-analysis first. Table 5.3-6 gives the number of best networks obtained for applying different combinations of initial network subroutines and TT-sequences. Obviously, the application of TT3, TT4, TT5 or TT6 tends to produce very good results, no matter what initial network subroutines are applied. The overall best results for each function and the combinations of an initial network subroutine and a TT-sequence from which the overall best results are derived are shown in Table 5.3-7. In order to get a clearer picture than Table 5.3-7, Table 5.3-8 and Table 5.3-9 are made. Table 5.3-8 gives the number of overall best results obtained when a specific initial network subroutine is applied. It indicates that among all best results, 22 can be derived from the initial networks obtained by BANDB. Table 5.3-9 gives the number of overall best results obtained when a specific TT-sequence is applied. This table shows that among all best results for 30 functions, 24 can

Table 5.3-2 Experimental results for 30 4-variable functions. Initial network subroutine UNIVSA is used.

COST & TIME / TT-SEQUENCE / FUNCTION (HEXADECIMAL)	COST						TIME(CS)					
	TT1	TT2	TT3	TT4	TT5	TT6	TT1	TT2	TT3	TT4	TT5	TT6
1. 4AF1	10023	10023	10023	10023	10023	10023	64	107	163	231	215	238
2. F6FE	8014	8014	8014	8014	8014	8014	36	85	104	123	144	147
3. AC6E	13024	13024	11021	11021	8021	8021	79	176	329	389	429	440
4. 2D86	16034	16034	14029	13028	13028	13028	162	624	378	799	923	1225
5. 9DA5	10022	10022	10022	6015	6015	6015	65	123	141	287	314	273
6. 5F12	7013	7013	7013	7013	7013	7013	50	74	90	118	103	113
7. F1F4	7014	7014	7014	7014	7014	7014	51	91	86	113	119	128
8. 6830	13027	11020	10020	9022	9022	8017	128	298	410	450	405	557
9. 9048	9022	9022	8023	8023	8023	8023	68	164	135	257	241	215
10. EA9B	9022	9022	9022	9022	9022	9022	59	104	131	171	164	184

(Continued)

Table 5.3-2 Experimental results for 30 4-variable functions.
Initial network subroutine UNIVSA is used.

COST & TIME TT-SEQUENCE FUNCTION (HEXADECIMAL)	COST						TIME(CS)					
	TT1	TT2	TT3	TT4	TT5	TT6	TT1	TT2	TT3	TT4	TT5	TT6
11. 68F5	10022	10022	10021	8019	8019	8019	76	280	252	284	281	207
12. 468F	12024	12024	11023	11023	11023	11023	77	183	265	375	376	360
13. B860	12026	11025	12026	10025	10025	10025	75	318	231	428	453	433
14. E139	13027	15029	11023	14029	14029	9023	185	732	419	949	1000	557
15. 70CF	6014	6014	6014	6014	6014	6014	42	58	66	86	90	91
16. F3DO	7013	7013	6011	6011	6011	6010	48	77	82	120	115	85
17. BC7A	12025	12025	12025	12025	12025	9021	73	164	196	282	539	188
18. F6AE	9019	9019	7017	7017	7017	7015	61	163	183	226	310	91
19. DE84	9020	12025	12025	12025	12025	10021	139	305	450	429	400	155
20. 6777	7012	7012	7012	7012	7012	7012	40	69	81	110	104	79

(Continued)

Table 5.3-2 Experimental results for 30 4-variable functions. Initial network subroutine UNIVSA is used.

COST & TT- TIME SEQUENCE / FUNCTION (HEXADECIMAL)	COST						TIME(CS)					
	TT1	TT2	TT3	TT4	TT5	TT6	TT1	TT2	TT3	TT4	TT5	TT6
21. C11B	8019	8019	9021	8019	8019	8019	131	312	169	329	310	234
22. C06F	6017	6017	6017	6017	6017	6017	50	66	63	96	87	99
23. D379	11023	13025	12025	12025	10023	12025	232	524	422	567	919	646
24. AE1F	6015	6015	6015	6015	6015	6015	67	94	76	109	98	117
25. A849	12027	13029	13029	13029	13029	11027	96	333	266	463	506	373
26. B896	15031	15029	14030	14030	14028	14029	153	481	454	882	742	521
27. 9BCB	10022	10022	6014	6014	8016	6014	65	155	157	239	467	334
28. 4710	8015	8014	9016	7012	7012	7012	71	181	104	221	203	233
29. 6433	6014	6014	6014	6014	6014	6014	47	64	72	97	84	96
30. 9903	6014	6014	6014	6014	6014	6014	52	77	77	95	93	105

Table 5.3-3 Experimental results for 30 4-variable functions. Initial network subroutine THRLEV is used.

COST & TIME TT-SEQUENCE FUNCTION (HEXADECIMAL)	COST						TIME(CS)					
	TT1	TT2	TT3	TT4	TT5	TT6	TT1	TT2	TT3	TT4	TT5	TT6
1. 4AF1	**10023**	**10023**	**10023**	**10023**	**10023**	**10023**	76	114	161	200	200	214
2. F6FE	**8014**	**8014**	**8014**	**8014**	**8014**	**8014**	44	62	66	97	106	112
3. AC6E	13024	13024	11021	11021	11021	**9020**	78	128	258	313	457	287
4. 2D86	19037	17032	19037	16032	13028	**13027**	134	679	555	1401	897	1515
5. 9DA5	10022	10022	10022	**6015**	**6015**	**6015**	58	108	128	245	288	265
6. 5F12	**7013**	**7013**	**7013**	**7013**	**7013**	**7013**	60	67	83	104	92	107
7. F1F4	8015	**7014**	**7014**	**7014**	**7014**	**7014**	59	145	83	96	101	112
8. 6830	11022	11022	13024	**9022**	**9022**	11022	106	207	273	328	306	289
9. 9048	9022	9022	**8023**	**8023**	**8023**	**8023**	49	143	124	221	198	166
10. EA9B	**9019**	**9019**	9022	9022	9022	**9019**	47	93	115	147	149	170

(Continued)

Table 5.3-3 Experimental results for 30 4-variable functions. Initial network subroutine THRLEV is used.

COST & TT-SEQUENCE TIME / FUNCTION (HEXADECIMAL)	COST						TIME(CS)					
	TT1	TT2	TT3	TT4	TT5	TT6	TT1	TT2	TT3	TT4	TT5	TT6
11. 68F5	10022	10022	10021	8019	8019	8019	71	222	198	261	258	195
12. 468F	12024	12024	12024	12024	12024	12024	68	140	170	274	277	291
13. B860	13026	12024	12026	10024	11025	12026	80	320	175	373	412	303
14. E139	13026	15030	11023	12026	12026	10022	181	512	388	798	835	369
15. 70CF	6014	6014	6014	6014	6014	6014	32	47	52	67	84	79
16. F3DO	7013	7013	6011	6011	6011	6011	47	80	79	111	148	117
17. BC7A	13025	13025	12025	12025	12024	9021	65	152	205	257	524	342
18. F6AE	10019	10019	7017	7017	7017	7017	60	98	142	175	183	204
19. DE84	11022	11022	12025	12025	9020	9020	66	131	234	407	621	320
20. 7012	7012	7012	7012	7012	7012	7012	29	60	46	75	74	88

(Continued)

Table 5.3-3 Experimental results for 30 4-variable functions. Initial network subroutine THRLEV is used.

COST & TIME / TT-SEQUENCE / FUNCTION (HEXADECIMAL)	COST						TIME(CS)					
	TT1	TT2	TT3	TT4	TT5	TT6	TT1	TT2	TT3	TT4	TT5	TT6
21. C11B	8019	8019	9021	8019	8019	8019	112	244	133	291	317	192
22. 606F	6017	6017	6017	6017	6017	6017	41	67	50	83	96	104
23. D379	13025	15032	12025	12025	12025	13025	191	549	442	532	857	510
24. AEIF	6015	6015	6015	6015	6015	6015	53	62	64	93	86	95
25. A849	13029	14030	12028	11027	11027	10024	99	204	217	505	517	341
26. B896	14029	14029	14030	13029	14028	11025	151	478	457	676	684	722
27. 9BCB	10022	10022	6014	8016	8016	8016	70	130	142	248	401	243
28. 5710	8014	8014	9016	7015	7015	7015	68	145	91	174	184	170
29. 6433	6014	6014	6014	6014	6014	6014	43	58	53	90	77	92
30. 9903	6014	6014	6014	6014	6014	6014	35	62	47	81	97	77

Table 5.3-4 Experimental results for 30 4-variable functions. Initial network subroutine BANDB is used.

COST & TIME / TT-SEQUENCE / FUNCTION (HEXADECIMAL)	COST						TIME(CS)					
	TT1	TT2	TT3	TT4	TT5	TT6	TT1	TT2	TT3	TT4	TT5	TT6
1. 4AF1	10020	10020	10020	[✓ 8017]	9021	9021	90	233	222	357	237	187
2. F6FE	[✓ 6012]	[✓ 6012]	[✓ 6012]	[✓ 6012]	[✓ 6012]	[✓ 6012]	34	50	48	62	72	75
3. AC6E	8016	8016	[✓ 7016]	[✓ 7016]	9018	9018	58	92	98	130	195	168
4. 2D86	15029	13026	13025	[✓ 11026]	14030	13027	227	659	569	969	938	905
5. 7DA5	8019	8019	[✓ 6015]	[✓ 6015]	[✓ 6015]	[✓ 6015]	47	83	114	128	139	127
6. 5F12	[✓ 7013]	[✓ 7013]	[✓ 7013]	[✓ 7013]	7015	7015	69	78	82	119	102	117
7. F1F4	[✓ 7014]	[✓ 7014]	[✓ 7014]	[✓ 7014]	[✓ 7014]	[✓ 7014]	36	64	62	80	72	89
8. 6830	13024	13022	13024	12022	[✓ 9022]	[✓ 9022]	86	232	255	404	280	246
9. 9048	9022	9022	[✓ 8023]	[✓ 8023]	11024	9021	77	170	144	262	325	449
10. EA9B	[✓ 9019]	[✓ 9019]	[✓ 9019]	[✓ 9019]	[✓ 9019]	[✓ 9019]	69	102	116	156	218	186

(Continued)

Table 5.3-4 Experimental results for 30 4-variable functions. Initial network subroutine BANDB is used.

COST & TT- TIME SEQUENCE FUNCTION (HEXADECIMAL)	COST						TIME(CS)					
	TT1	TT2	TT3	TT4	TT5	TT6	TT1	TT2	TT3	TT4	TT5	TT6
11. 68F5	8020	8020	8020	8020	8020	8020	38	65	79	113	123	130
12. 4685	12024	11021	9019	9019	10023	8017	80	266	346	365	446	273
13. B860	11021	13025	13024	13025	13025	9019	134	353	398	458	468	408
14. E139	11026	11026	11026	11026	14030	14030	84	154	185	250	505	416
15. 70CF	6013	6013	6013	6013	6013	6013	54	51	60	76	74	84
16. F3DO	6010	6010	6010	6010	6010	6010	47	65	50	76	87	85
17. BC7A	9020	9020	8020	8020	10022	9021	56	101	134	168	247	188
18. F6AE	7015	7015	7015	7015	7015	7015	42	61	70	79	81	91
19. DE84	10021	8016	10021	8016	8016	1021	49	214	123	211	222	155
20. 6777	7012	7012	7012	7012	7012	7012	32	55	52	76	78	79

(Continued)

Table 5.3-4 Experimental results for 30 4-variable functions.
Initial network subroutine BANDB is used.

COST & TIME TT-SEQUENCE FUNCTION (HEXADECIMAL)	COST						TIME(CS)					
	TT1	TT2	TT3	TT4	TT5	TT6	TT1	TT2	TT3	TT4	TT5	TT6
21. C11B	10023	10023	10023	10023	10023	8020	70	138	135	197	212	292
22. C06F	10019	10019	7014	7014	7014	7014	62	110	188	236	249	268
23. D379	11025	11025	11025	11025	11025	11025	99	240	224	358	345	282
24. AE1F	6015	6015	6015	6015	6015	6015	46	49	56	74	81	83
25. A849	11023	11023	15029	10022	11023	10022	150	234	690	526	630	638
26. B896	15030	14025	14031	13027	14029	15031	212	515	494	698	860	1273
27. 9BCB	7014	7014	6014	6014	6014	6014	70	116	80	103	95	115
28. 5710	7013	7013	7013	7013	7013	7013	59	80	81	98	146	116
29. 6433	6013	6013	6013	6013	6013	6013	44	55	52	72	67	78
30. 9903	6014	6014	6014	6014	6014	6014	46	55	53	66	68	86

Table 5.3-5 Experimental results for 30 4-variable functions. Initial network subroutine TISON is used.

COST & TIME TT-SEQUENCE FUNCTION (HEXADECIMAL)	COST						TIME(CS)					
	TT1	TT2	TT3	TT4	TT5	TT6	TT1	TT2	TT3	TT4	TT5	TT6
1. 4AF1	11022	11022	10022	10022	10022	10022	58	114	173	217	206	213
2. F6FE	8014	8014	8014	8014	8014	8014	33	56	74	91	91	94
3. AC6E	9019	9019	9019	9019	9019	9019	80	209	189	242	237	183
4. 2D86	14029	16029	17031	12024	12024	13029	176	560	461	1014	1002	624
5. 9DA5	11021	11021	10022	10021	10021	10021	59	174	114	258	247	230
6. 5F12	7013	7013	7013	7013	7013	7013	41	52	60	80	80	79
7. F1F4	7014	7014	7014	7014	7014	7014	32	55	60	75	79	90
8. 6830	12023	12023	11023	11023	11023	11023	62	132	160	222	202	226
9. 9048	15028	15028	15028	15028	15028	11022	62	196	250	400	392	773
10. EA9B	9019	9019	9019	9019	9019	9019	59	91	112	146	143	151

(Continued)

Table 5.3-5 Experimental results for 30 4-variable functions.
Initial network subroutine TISON is used.

COST & TIME / TT-SEQUENCE / FUNCTION (HEXADECIMAL)	COST						TIME(CS)					
	TT1	TT2	TT3	TT4	TT5	TT6	TT1	TT2	TT3	TT4	TT5	TT6
11. 68F5	13023	10020	13027	11025	8020	10022	97	384	348	540	529	573
12. 468F	12025	12026	12026	12026	10022	9020	80	125	150	249	433	329
13. B860	12024	12024	12024	11024	12024	12024	64	123	148	243	236	251
14. E139	14027	14027	10024	11024	11024	9021	99	424	461	775	764	396
15. 70CF	7014	7014	6014	6014	6014	6014	35	50	51	76	69	85
16. F3DO	7013	7013	6011	6011	6011	6011	36	57	65	96	81	92
17. FC7A	13025	13025	12025	12025	12025	12025	52	151	161	254	271	279
18. F6AE	10019	10019	7017	7017	7017	7017	40	84	129	179	169	181
19. DE84	10017	10017	11022	9017	9017	9017	75	239	108	304	287	190
20. 6777	7012	7012	7012	7012	7012	7012	34	65	51	80	71	87

(Continued)

Table 5.3-5 Experimental results for 30 4-variable functions. Initial network subroutine TISON is used.

COST & TIME / TT-SEQUENCE / FUNCTION (HEXADECIMAL)	COST						TIME(CS)					
	TT1	TT2	TT3	TT4	TT5	TT6	TT1	TT2	TT3	TT4	TT5	TT6
21. C11B	10021	11023	9021	9021	9021	9021	103	238	152	257	258	239
22. C06F	7015	7015	7015	7015	7015	7015	47	77	82	91	94	101
23. D379	12025	12025	11023	11023	11023	12025	158	402	746	582	548	396
24. AE1F	7015	7015	6015	6015	6015	6015	43	69	56	89	84	89
25. A849	12025	12025	11024	11024	11024	11024	68	139	165	248	248	264
26. B896	12024	14029	17034	13030	16033	15031	275	675	624	880	1253	602
27. 9BCB	9016	7014	8016	8016	8016	8016	66	215	135	191	185	202
28. 5710	7014	7014	9016	7014	7014	7014	66	130	83	152	154	116
29. 6433	7014	7014	6014	6014	6014	6014	48	77	82	108	88	105
30. 9903	7014	7014	6014	6014	8016	6014	52	68	83	91	168	121

Table 5.3-6 Numbers of best networks obtained by different combinations of initial network subroutines and TT-sequences

INITIAL NETWORK SUBROUTINE	TT-SEQUENCE					
	TT1	TT2	TT3	TT4	TT5	TT6
UNIVSA	13	12	15	20	22	27
THRLEV	12	13	16	21	21	26
BANDB	15	16	21	25	18	21
TISON	8	9	19	25	23	24

Table 5.3-7 Overall best results and the combinations of initial network subroutines and TT-sequences from which the best overall results are obtained for each function

FUNC. NO.	Best Cost	Combination of initial network subroutines and TT-sequences	FUNC. NO.	Best Cost	Combination of initial network subroutines and TT-sequences
1	8017	$\binom{\text{BANDB}}{\text{TT4}}$	16	6010	$\binom{\text{UNIVSA}}{\text{TT6}}$ $\binom{\text{BANDB}}{\text{TT1-TT6}}$
2	6012	$\binom{\text{BANDB}}{\text{TT1-TT6}}$	17	8020	$\binom{\text{BANDB}}{\text{TT3,TT4}}$
3	7016	$\binom{\text{BANDB}}{\text{TT3-TT4}}$	18	7015	$\binom{\text{UNIVSA}}{\text{TT6}}$ $\binom{\text{BANDB}}{\text{TT1-TT6}}$
4	11026	$\binom{\text{BANDB}}{\text{TT4}}$	19	8016	$\binom{\text{BANDB}}{\text{TT2,TT4,TT5}}$
5	6015	$\binom{\text{UNIVSA}}{\text{TT4-TT6}}$ $\binom{\text{THRLEV}}{\text{TT4-TT6}}$ $\binom{\text{BANDB}}{\text{TT3-TT6}}$	20	7012	ALL COMBINATIONS
6	7013	ALL EXCEPT $\binom{\text{BANDB}}{\text{TT5-TT6}}$	21	8019	$\binom{\text{UNIVSA}}{\text{EXCEPT TT3}}$ $\binom{\text{THRLEV}}{\text{EXCEPT TT3}}$
7	7014	ALL EXCEPT $\binom{\text{THRLEV}}{\text{TT1}}$	22	6017	$\binom{\text{UNIVSA}}{\text{TT1-TT6}}$ $\binom{\text{THRLEV}}{\text{TT1-TT6}}$
8	8017	$\binom{\text{UNIVSA}}{\text{TT6}}$	23	10023	$\binom{\text{UNIVSA}}{\text{TT5}}$
9	8023	$\binom{\text{UNIVSA}}{\text{TT4-TT6}}$ $\binom{\text{THRLEV}}{\text{TT3-TT6}}$ $\binom{\text{BANDB}}{\text{TT3-TT4}}$	24	6015	ALL EXCEPT $\binom{\text{TISON}}{\text{TT1,TT2}}$
10	9019	$\binom{\text{THRLEV}}{\text{TT1,TT2,TT6}}$ $\binom{\text{BANDB}}{\text{TT1-TT6}}$ $\binom{\text{TISON}}{\text{TT1-TT6}}$	25	10022	$\binom{\text{BANDB}}{\text{TT4,TT6}}$
11	8019	$\binom{\text{UNIVSA}}{\text{TT4-TT6}}$ $\binom{\text{THRLEV}}{\text{TT4-TT6}}$	26	11025	$\binom{\text{THRLEV}}{\text{TT6}}$
12	8017	$\binom{\text{BANDB}}{\text{TT6}}$	27	6014	$\binom{\text{UNIVSA}}{\text{TT3,TT4,TT6}}$ $\binom{\text{THRLEV}}{\text{TT6}}$ $\binom{\text{BANDB}}{\text{TT3-TT6}}$
13	9019	$\binom{\text{BANDB}}{\text{TT6}}$	28	7012	$\binom{\text{UNIVSA}}{\text{TT4-TT6}}$
14	9021	$\binom{\text{TISON}}{\text{TT6}}$	29	6013	$\binom{\text{BANDB}}{\text{TT1-TT6}}$
15	6031	$\binom{\text{BANDB}}{\text{TT1-TT6}}$	30	6014	ALL EXCEPT $\binom{\text{TISON}}{\text{TT1,TT2,TT5}}$

be derived if either TT-sequence TT4 or TT-sequence TT6 is applied. These results mean that in order to get results with very good costs, the combinations of BANDB-TT4 or BANDB-TT6 are more desirable.

Table 5.3-8 Number of overall best results obtained when a specific initial network subroutine is applied

INITIAL NETWORK SUBROUTINES INVOLVED	No. of overall best results derived
UNIVSA	16
THRLEV	13
BANDB	22
TISON	7

Table 5.3-9 Number of overall best results obtained when a specific initial network TT-sequence is applied

TT-SEQUENCE APPLIED	No. of overall best results derived
TT1	13
TT2	14
TT3	17
TT4	24
TT5	20
TT6	24

Another interesting statistic is the average cost per function for different combinations of initial network subroutines and TT-sequences. This is shown in Table 5.3-10. Again, we find from this table that the combination of BANDB and TT4 produces the best average cost, and the combinations involving BANDB and/or TT4 or TT6 usually produce networks with better average costs.

Table 5.3-10 Average costs per function for different combinations of initial network subroutines and TT-sequences

INITIAL NETWORK SUBROUTINE	TT-SEQUENCE					
	TT1	TT2	TT3	TT4	TT5	TT6
UNIVSA	9.60 /20.47*	9.73 /20.40	9.10 /19.77	8.97 /19.47	8.73 /19.33	8.30 /18.63
THRLEV	9.87 /20.40	9.90 /20.47	9.37 /20.17	8.87 /19.57	8.77 /19.23	8.47 /18.67
BANDB	8.98 /18.50	8.80 /18.17	8.60 /18.47	8.00 /17.80	8.67 /18.63	8.33 /18.10
TISON	9.83 /18.50	9.47 /19.43	9.77 /19.83	9.17 /19.13	9.20 /19.00	8.93 /18.87

*The average cost is expressed as $^{R_{av}}/_{C_{av}}$, where R_{av} and C_{av} are the average number of gates and connections, respectively.

So far the analyses are all for costs. For comparing computation times, other two tables are shown below. Table 5.3-11 gives the average computation time per function for different combinations of initial network subroutines and TT-sequences. Besides, we can get the ratio of computation times by using the time spent by the combination UNIVSA and TT1 as the base. This result is shown in Table 5.3-12.

Table 5.3-11 Average computation time per function in centiseconds for different combinations of initial network subroutines and TT-sequences.

INITIAL NETWORK SUBROUTINE	TT-SEQUENCE					
	TT1	TT2	TT3	TT4	TT5	TT6
UNIVSA	30.01	215.43	199.17	307.80	338.33	284.13
THRLEV	76.13	183.57	220.90	290.77	317.50	256.27
BANDB	75.57	158.00	175.33	232.23	255.40	256.30
TISON	73.33	181.07	184.33	274.33	288.97	245.37

Table 5.3-12 Rate of computation times. The time spent by the combinaion UNIVSA and TT1 is used as the base.

INITIAL NETWORK SUBROUTINE	TT-SEQUENCE					
	TT1	TT2	TT3	TT4	TT5	TT6
UNIVSA	1	7.18	6.64	10.26	11.28	9.47
THRLEV	2.54	6.12	7.36	9.69	10.58	8.54
BANDB	2.51	5.27	5.84	7.74	8.51	8.54
TISON	2.44	6.03	6.14	9.14	9.62	8.17

Table 5.3-11 and Table 5.3-12 indicate that TT-sequence TT1 is less time-consuming than other TT-sequences whereas TT-sequences TT4, TT5 and TT6 are more time-consuming.

The experimental results for 5-variable functions are analyzed in a similar manner. Table 5.3-13 through Table 5.3-16 give network costs and computation times. The best results for each function are marked with circles in each table. The numbers of best networks obtained by different combinations of initial network subroutines and TT-sequences are given in Table 5.3-17. The overall best results and the combinations of initial network subroutines and TT-sequences from which the overall results are derived are given in Table 5.3-18. Again, the number of overall best results obtained when a specific initial network subroutine and a specific TT-sequence is applied are given in Table 5.3-19 and Table 5.3-20, respectively.

Table 5.3-13 Experimental results for 10 5-variable functions. Initial network subroutine UNIVSA is used.

COST & TT- TIME SEQUENCE / FUNCTION (HEXADECIMAL)	COST						TIME(CS)					
	TT1	TT2	TT3	TT4	TT5	TT6	TT1	TT2	TT3	TT4	TT5	TT6
1. 4FA295F6	19047	17044	15040	15040	19043	19043	319	1187	1619	2216	4357	4127
2. A6CDDF18	17042	17042	16043	15039	14040	16038	599	1475	2635	2672	3318	2485
3. FF68A1F3	13034	13034	14036	12036	12036	12036	283	763	682	1348	1349	1328
4. 1EE65240	15035	15035	16035	15035	15035	13033	245	869	2292	1541	1458	1255
5. 9E63EE75	18041	15037	14032	12033	14031	15035	382	1113	2141	2350	2820	2307
6. 0A88103	12031	11024	11033	11033	11033	11033	207	675	356	523	498	542
7. 49F363CD	17045	16044	16044	16044	17045	17046	237	909	723	1074	3171	2088
8. 8B5809F0	12031	13031	12033	11031	10031	11031	302	1056	725	1603	1159	1422
9. BFDBC6DA	17042	15040	16042	14040	14040	18044	587	1238	2312	2174	3274	2856
10. C6E7103E	16039	16039	16040	15040	16040	16040	261	1015	928	1853	1303	1253

Table 5.3-14 Experimental results for 10 5-variable functions. Initial network subroutine THRLEV is used.

COST & TIME TT-SEQUENCE FUNCTION (HEXADECIMAL)	COST						TIME(CS)					
	TT1	TT2	TT3	TT4	TT5	TT6	TT1	TT2	TT3	TT4	TT5	TT6
1. 4FA295F6	20049	17044	16041	15040	19045	19045	359	1295	1393	2197	2920	2578
2. A6CDDF18	17040	17038	21050	16040	16038	16039	472	1374	1342	3420	2667	2702
3. FF68A1F3	16038	13034	14038	12036	12036	12036	210	671	1316	1171	1150	985
4. 1EE65240	15035	15035	15036	15035	15035	13033	159	607	1183	1153	2258	882
5. 9E63BE75	15037	14037	14032	13032	13032	12031	312	1024	1801	2026	1256	1119
6. 0A888103	12031	11024	11033	11033	11033	11033	135	441	251	358	357	395
7. 49F363CD	17044	16043	16044	16044	14036	17046	277	942	696	999	1688	1885
8. 8B5809F0	15035	13031	11030	11031	10031	11031	203	690	1177	1097	896	756
9. BFDBC6DA	17041	18044	16039	17040	14034	13033	509	1339	3094	2956	3280	2428
10. C6E7103E	16041	14039	14039	14039	16039	14036	362	759	622	1020	2869	1690

Table 5.3-15 Experimental results for 10 5-variable functions. Initial network subroutine BANDB is used.

COST & TIME TT-SEQUENCE FUNCTION (HEXADECIMAL)	COST						TIME(CS)					
	TT1	TT2	TT3	TT4	TT5	TT6	TT1	TT2	TT3	TT4	TT5	TT6
1. 4FA295F6	15038	15037	15039	15037	15038	15037	385	868	733	1355	2094	1210
2. A6CDDF18	19050	18048	16047	18047	17042	18045	391	1355	1081	1906	3340	1794
3. FF68A1F3	14032	13030	12034	11033	13034	11031	846	1270	1888	2600	3576	3095
4.. 1EE65240	13031	17037	11030	11030	16041	16043	349	702	1027	1582	2258	1317
5. 9E63BE75	12034	12034	12035	12034	13036	13036	385	670	521	909	1256	1528
6. 0A888103	10026	10026	9026	9026	9026	11032	191	263	330	409	487	652
7. 49F363CD	10032	10032	11032	11032	11031	11031	263	398	444	568	885	1185
8. 8B5809F0	13036	13036	12033	12033	12029	11031	190	316	495	691	776	1194
9. BFDBC6DA	17044	17041	17044	16041	17043	17041	194	713	603	1359	1302	1617
10. C6E7103E	13034	13034	13034	13034	13034	13034	219	544	534	671	863	675

Table 5.3-16 Experimental results for 10 5-variable functions. Initial network subroutine TISON is used.

COST & TIME / TT-SEQUENCE / FUNCTION (HEXADECIMAL)	COST						TIME(CS)					
	TT1	TT2	TT3	TT4	TT5	TT6	TT1	TT2	TT3	TT4	TT5	TT6
1. 4FA295F6	19043	19042	22051	17042	17042	17042	283	1065	1202	1877	1961	1946
2. A6CDDF18	21050	18045	21050	18045	18045	18045	183	1433	1083	2189	2276	2319
3. FF68A1F3	17044	17044	15040	17044	17041	17041	121	359	619	851	1073	1158
4. 1EE65240	15037	15037	15037	15037	15037	15037	176	335	513	719	731	837
5. 9E63BE75	15038	15038	15038	15038	15038	15038	94	322	397	633	670	788
6. 0A888103	12027	12027	11028	12027	12027	12027	195	380	624	686	810	633
7. 49F363CD	17039	16039	16044	16044	17044	17044	511	2290	628	860	1941	2032
8. 8B5809F0	14037	14037	14037	14037	14037	14037	267	437	627	972	800	842
9. BFDBC6DA	22054	17042	22054	17042	17042	18044	141	1695	1505	2450	2516	3504
10. C6E7103E	14039	14039	14039	14039	14039	14039	128	310	388	577	580	617

Table 5.3-17 Numbers of best results obtained by different combinations of initial network subroutines and TT-sequences.

INITIAL NETWORK SUBROUTINE	TT-SEQUENCES					
	TT1	TT2	TT3	TT4	TT5	TT6
UNIVSA	0	1	3	7	5	3
THRLEV	0	1	0	2	4	5
BANDB	2	3	4	6	3	5
TISON	4	7	6	7	7	6

Table 5.3-18 Overall best results

5-VAR. FUNCTION NO.	OVERALL BEST COST	COMBINATIONS OF INITIAL NETWORK SUBROUTINES AND TT-SEQUENCES WHICH YIELD OVERALL BEST 24 RESULTS
1	15037	(BANDB TT2,TT4,TT6)
2	14040	(UNIVSA TT5)
3	11031	(BANDB TT6)
4	11030	(BANDB TT3,TT4)
5	12031	(THRLEV TT6)
6	9026	(BANDB TT3,TT4,TT5)
7	11031	(BANDB TT5,TT6)
8	10031	(UNIVSA TT5) (THRLEV TT5)
9	13033	(THRLEV TT6)
10	13034	(BANDB TT1-TT6)

The average cost per function for different combinations of initial network subroutines and TT-sequences is shown in Table 5.3-21. From the previous tables for analyzing network costs for 5-variable functions, we get the similar conclusions as in 4-variable case; that is, the combinations of BANDB-TT4, BANDB-TT5, or BANDB-TT6 usually produce networks with very good costs.

Table 5.3-19 Numbers of overall best results obtained when a specific initial network subroutine is applied.

INITIAL NETWORK SUBROUTINE	NO. OF OVERALL BEST RESULTS OBTAINED
UNIVSA	2
THRLEV	3
BANDB	6
TISON	0

Table 5.3-20 Numbers of overall best results obtained when a specific TT-sequence is applied.

TT-SEQUENCE	NO. OF OVERALL BEST RESULTS OBTAINED
TT1	1
TT2	2
TT3	3
TT4	4
TT5	5
TT6	6

Table 5.3-21 Average cost per function[*] for different combinations of initial network subroutines and TT-sequences.

INITIAL NETWORK SUBROUTINE	TT-SEQUENCE					
	TT1	TT2	TT3	TT4	TT5	TT6
UNIVSA	15.6 /38.7	14.8 /37.0	14.6 /37.8	13.6 /37.1	14.2 /37.3	14.8 /38.4
THRLEV	16.1 /39.1	14.8 /36.9	14.8 /38.2	14.0 /37.0	14.0 /35.9	13.8 /36.3
BANDB	13.7 /35.7	13.9 /35.5	12.8 /35.4	12.8 /34.7	13.6 /38.4	13.6 /36.1
TISON	16.6 /40.8	15.7 /39.0	16.5 /42.2	15.5 /39.9	15.6 /39.4	15.7 /39.4

[*] Average numbers of gates and connections are shown by upper and lower figures, respectively, in each cell.

The computation time is analyzed in Table 5.3-22 and Table 5.3-23. Table 5.3-22 shows the average computation times for different combinations of initial network subroutines and TT-sequences whereas Table 5.3-23 gives the ratios by using the time spent by UNIVSA and TT1 as the base. Again, we get the similar conclusions as the 4-variable case: TT1 is the least time-consuming TT-sequence whereas TT5 and TT6 are usually more time-consuming than other TT-sequences.

Table 5.3-22 Average computation times in centiseconds for different combinations of initial network subroutines and TT-sequences.

INITIAL NETWORK SUBROUTINE	TT-SEQUENCE					
	TT1	TT2	TT3	TT4	TT5	TT6
UNIVSA	342.2	1030.0	1411.3	1735.4	2270.7	1966.3
THRLEV	299.8	914.2	1287.4	1639.7	1834.1	1542.0
BANDB	341.3	709.9	765.6	1205.0	1683.7	1426.7
TISON	209.9	862.6	758.6	1181.4	1335.8	1467.6

Table 5.3-23 Ratios of computation times. The time spent by the combination of UNIVSA and TT1 is used as the base.

INITIAL NETWORK SUBROUTINE	TT-SEQUENCE					
	TT1	TT2	TT3	TT4	TT5	TT6
UNIVSA	1	3.01	4.12	5.07	6.64	5.75
THRLEV	0.88	2.67	3.76	4.79	5.36	3.41
BANDB	0.99	2.07	2.21	3.51	4.92	4.17
TISON	0.61	2.52	2.22	3.45	3.90	4.29

After analyzing the effectiveness and efficiency of each TT-sequence, we have some ideas on how to specify a control sequence to solve given problems under fan-in/fan-out restrictions. But for the users who are not interested in the details about how to specify control sequences, three control sequences provided in the NETTRA system can be used. These control sequences are called control sequences OPTION 1, OPTION 2 and OPTION 3. Users only need to specify the type of these options. The details about setting up data cards for using the NETTRA system is explained in [13].

The contents of three control sequences are listed in Table 5.3-24. OPTION 1 is a control sequence which can usually produce networks with very good costs (TT-sequence TT4 is used), OPTION 2 is a control sequence which can produce networks with reasonably good costs in a reasonably short computation time (TT-sequence TT3 is used), and OPTION 3 is a control sequence which can produce networks in a very short time. Some practical problems are used to test these built-in control sequences. Table 5.3-25 shows the results. Apparently, these results satisfy our requirements. For Su-Nam's multiple-output incompletely specified example, the network obtained by the NETTRA system using the control sequence OPTION 1 consists of 6 levels, 15 gates and 29 connections, while the original result obtained by Su and Nam consists of 6 levels, 25 gates and 42 connections [39]; i.e., remarkable reduction in the cost has been obtained.

Table 5.3-24 Contents of three built-in control sequences

TYPE OF CONTROL SEQUENCE	TYPE OF INITIAL NETWORK SUBROUTINES	TYPE OF TT-SEQUENCE	NO. OF TIMES THAT TT-SEQUENCE IS GOING TO BE EXECUTED
OPTION 1	UNIVSA, THRLEV and BANDB	TT4	∞
OPTION 2	BANDB	TT3	∞
OPTION 3	UNIVSA	TT4	1

For the 3-variable odd parity function and the three-variable even parity function, control sequences OPTION 1 and OPTION 2 can derive networks with 8 gates and 16 connections and 7 gates and 16 connections, respectively. These networks are optimal under the specified restrictions. The optimality of these results is proved by the integer programming logic design program based on the branch-and-bound method [25]. It is known that for the four-variable odd and even parity functions, the optimal networks under the specified restrictions have 10 gates and 29 connections and 10 gates and 24 connections, respectively [18]. Control sequences OPTION 1 and OPTION 2 produce the optimal result for the 4-variable odd parity function. For the one-bit full adder, control sequence OPTION 1 produces the optimal network under the specified restrictions in 6.50 seconds. The optimality of this network is proved by the integer programming logic design program based on the implicit enumeration method, spending 154.34 seconds [25], and also by the integer programming logic design program based on the branch-and-bound method, spending 14.80 seconds.

Table 5.3-25 Results for testing three built-in control sequences

PROBLEMS	RESTRICTIONS	OPTION 1		OPTION 2		OPTION 3	
		COST	TIME (CS)	COST	TIME (CS)	COST	TIME (CS)
3-VAR. ODD PARITY FUNCTION	FI = FO = FOX = FOO = 3 UC[*] ONLY	8016	839	8017	110	10020	73
3-VAR. EVEN PARITY FUNCTION		7017	718	7016	139	10021	42
4-VAR. ODD PARITY FUNCTION	FI = FO = FOX = FOO = 4 UC ONLY	10029	6316	10029	1096	13033	323
4-VAR. EVEN PARITY FUNCTION		12040	5577	12036	767	14033	455
5-VAR. ODD PARITY FUNCTION	FI = FO = FOX = FOO = 5 UC ONLY	18044	28499	18054	5871	22053	4427
5-VAR. EVEN PARITY FUNCTION		16041	49013	21061	21886	28072	2641
1-BIT FULL ADDER		8023	650	9018	183	9025	37
2-BIT FULL ADDER		16042	24514	18043	6680	23062	2646
2-BIT MULTIPLIER		12025	2826	12025	885	12027	146
SU-NAM's EXAMPLE	FI = FO = FOX = FOO = 2 C[†] AND UC	15029	11931	30047	3264	18032	404

[*] UC: uncomplemented external variables

[†] C: complemented external variables

Only problems under fan-in/fan-out restrictions are tested above. Next let us test problems under both fan-in/fan-out and level restrictions. A control sequence OPTION 4 is provided to treat problems according to the flowchart shown in Fig. 4.2-2. Control sequence OPTION 4 consists of initial network subroutine TISLEV and five transduction subroutines, as shown in Table 5.3-26. In this case, each transduction subroutine is applied under both fan-in/fan-out and level restrictions.

Table 5.3-26 Contents of the control sequence OPTION 4

INITIAL NETWORK SUBROUTINE	TT-SEQUENCE*	NO. OF TIMES OF APPLICATION OF TT-SEQUENCE
TISLEV	SUBSTI ∞, GTMERG ∞, PRIIFF ∞, PROCIV ∞, PROCCE ∞	1

*Each transduction subroutine in this TT-sequence is to be applied under fan-in/fan-out and level restrictions, and each transduction subroutine will be applied repeatedly until there is no further improvement in the cost.

It was mentioned in Chapters 2 and 4 that the approach shown in Fig. 4.2-2 does not guarantee that a network satisfying the given restrictions can be obtained even if there do exist feasible networks. However, the experimental results in Table 5.3-27 and Table 5.3-28 show that this approach is reasonably good. In Table 5.3-27, ten 4-variable functions are tested; the fan-in/fan-out restrictions are set as FI = FO = FOX = FOO = 3, the maximum number of level permitted (denoted by LEVLIM) is 4 and 5, and only uncomplemented external variables are permitted as inputs. When LEVLIM = 4, seven feasible networks are obtained; and when LEVLIM = 5, feasible networks are obtained for all functions. In Table 5.3-28, ten 5-variable functions are tested; the fan-in/fan-out restrictions are set as FI = FO = FOX = FOO = 4, the maximum number of levels permitted is 4 and 5, and both complemented and uncomplemented external variables are permitted as inputs. When either LEVLIM = 4 or LEVLIM = 5, we obtain feasible networks for all functions.

The results obtained for 10 5-variable functions are also compared with the results obtained by the integer programming logic design program based on the branch-and-bound method in Table 5.3-29. The integer programming logic design program based on the branch-and-bound method cannot produce any optimal networks within two minutes. It only produces two intermediate networks; and for Function 7 the intermediate result is even worse than the result obtained by the control sequence OPTION 4.

Table 5.3-27 Experimental results for testing the control sequence OPTION 4

FUNCTION (4-VAR.)	FI = FO = FOX = FOO = 3; uncomplemented inputs only			
	LREST = 4		LREST = 5	
	COST	TIME (CS)	COST	TIME (CS)
1. 4AF1	8021	167	8021	182
2. FBFE	8014	111	8014	123
3. ABCE	7017*	504	7017	274
4. 2D8B	13026*	922	13026	405
5. 9DA5	7018	116	6015	117
6. 5F12	7013	95	7013	85
7. F1F4	7014	85	7014	84
8. 6830	11021	288	8019	225
9. 9048	14026*	687	14026	425
10. EA9B	9019	167	9019	183

*These results are not level-restricted.

Table 5.3-28 Experimental results for testing the control sequence OPTION 4

FUNCTION (5-VAR.)	FI = FO = FOX = FOO = 3;		complemented and uncomplemented inputs	
	LEVLIM = 4		LEVLIM = 5	
	COST	TIME (CS)	COST	TIME (CS)
1. 4FA295F6	19043	928	17041	2660
2. A6CDDF18	16040	996	15036	957
3. FF68A153	13033	1257	14034	1173
4. 1EE65240	12030	577	12030	716
5. 9E63BE75	16036	1251	16036	1077
6. 0A888103	11026	504	9021	375
7. 49F363CD	14033	1641	12030	1738
8. 8B5809F0	11028	756	10026	595
9. BFD6C6DA	14036	4485	14036	2494
10. C6E7103E	13034	1199	13032	1602

Table 5.3-29 Comparison between the NETTRA system using OPTION 4 and the program based on the branch-and-bound method.

RESTRICTIONS
COST & TIME
FUNCTIONS (HEX)
	LREST = 4
	FI = FO = FOX = FOO = 3;
		complemented and
		uncomplemented inputs

	RESULTS BY THE TRANSDUCTION PROGRAM	RESULTS BY THE BRANCH-AND-BOUND METHOD
	COST & TIME (CS)	COST & TIME (CS)
1. 4FA295F6	19043 928 CS	NO RESULTS AFTER TWO MINUTES
2. A6CDDF18	16040 996 CS	
3. FF68A153	13033 1257 CS	
4. 1EE65240	12030 577 CS	
5. 9E63BE75	16036 1251 CS	INTERMEDIATE RESULTS 14038 12000 CS
6. 0A888103	11026 504 CS	NO RESULTS AFTER TWO MINUTES
7. 49F363CD	14033 1641 CS	INTERMEDIATE RESULTS 15039 12000 CS
8. 8B5809F0	11028 756 CS	NO RESULTS AFTER TWO MINUTES
9. BFD6C6DA	14036 4485 CS	
10. C6E7103E	13034 1197 CS	

6. Conclusions

In this paper, the organization of the entire NETTRA system is introduced. Many experiments have been made to find out the effectiveness of the NETTRA system.

The current version of the NETTRA system can design near optimal, multiple-output, multi-level and loop-free NOR(NAND)-gate networks under fan-in/fan-out restrictions and/or level restriction. The problem size can be up to five external variables and ten output functions; each function may be completely or incompletely specified and both complemented and uncomplemented external variables or only uncomplemented external variables are allowed as inputs. Since all programming variables in the current version of the NETTRA system are 4-character integers, the system can be modified to treat problems with larger problem size by using 2-character integers as programming variables.

The experimental results given in this paper indicate that the NETTRA system is very effective in finding near-optimal solutions for the given problems. For switching functions with very few number of gates (around 9 through 12 gates), the NETTRA system can often produce networks which are really optimal. For the switching functions with many gates, the NETTRA systen cam produce networks with reasonably good costs in a short time that usually the integer programming logic design programs based on the branch-and-bound method and the implicit enumeration method cannot get any feasible solution.

Comparing with other methods such as transformation methods, the NETTRA system is generally much more powerful. Hence the NETTRA system appears to be a very useful and powerful tool for logic design of large NOR(NAND) networks.

The explanation of how to use the system is given in the programming manual [13].

The network transduction program NETTRA-E3 is not included in the NETTRA system because this program requires too much core storage. Since this program realizes the "multiple-path" application of the error-compensation procedures, it usually produces very good results if enough computation time is allowed. How to use this program is detailed in [21].

References

[1] Culliney, J.N., "Program manual: NOR network transduction based on connectable and disconnectable conditions (Reference manual of NOR network transduction programs NETTRA-G1 and NETTRA-G2)," UIUCDCS-R-75-698, Dept. of Computer Science, Univ. of Illinois, Urbana, Ill., Feb. 1975.

[2] Culliney, J.N., H.C. Lai, and Y. Kambayashi, "Pruning procedures for NOR networks using permissible functions (Principles of NOR network transduction programs NETTRA-PG1, NETTRA-P1, and NETTRA-P2), UIUCDCS-R-74-690, Dept. of Computer Science, Univ. of Illinois, Urbana, Ill., Nov. 1974.

[3] Cutler, R.B., et al, "TISON-VERSION 1: A program to derive a minimal sum of products for a function or set of functions which may not be completely specified," to appear as Report. Department of Computer Science, University of Illinois, Urbana, Ill.

[4] Davidson, E.S., "An Algorithm for NAND Decomposition of Combinational Switching Systems", Ph.D. dissertation, Department of Electrical Engineering and Coordinated Science Laboratory, Univ. of Illinois, Urbana, Ill., 1968.

[5] Dietmeyer, D.L., and Y.H. Su, "Logic design automation of fan-in limited NAND networks," IEEETC, Vol. C-18, No. 1, Jan. 1969.

[6] Ellis, D.T., "A synthesis of combinational logic with NAND or NOR elements," IEEE Trans., vol. EC-14, pp. 701-705, Oct. 1965.

[7] Gimpel, J.F., "The minimization of TANT networks," IEEE Trans., vol. EC-16, pp. 18-36, Feb. 1967.

[8] Hellerman, L., "A catalog of three-variable OR-INVERT and AND-INVERT logical circuits," IEEE Trans., vol. EC-12, pp. 198-223, June 1963.

[9] Hohulin, K.R., "Network transduction programs based on connectable and disconnectable conditions with fan-in and fan-out restrictions (A description of NETTRA-G1-FIFO and NETTRA-G2-FIFO)," UIUCDCS-R-75-719, Dept. of Computer Science, Univ. of Illinois, Urbana, Ill., April 1975.

[10] Hohulin, K.R., and S. Muroga, "Alternative methods for solving the c-c table in Gimpel's algorithm for synthesizing optimal three level NAND networks," UIUCDCS-R-75-720, Dept. of Computer Science, Univ. of Illinois, Urbana, Ill., April 1975.

[11] Hu, K.C., "NOR(NAND) network design; error-compensation procedures for fan-in/fan-out restricted networks (NETTRA-E1-FIFO and NETTRA-E2-FIFO)," Report No. UIUCDCS-R-77-847, Dept. of Computer Science, University of Illinois, Urbana, Ill.

[12] Hu, K.C., "Level-restricted NOR network transduction procedures," Report No. UIUCDCS-R-77-849, Dept. of Computer Science, University of Illinois, Urbana, Ill.

[13] Hu, K.C., "Programming manual for the NOR network transduction system," to appear as Report. Dept. of Computer Science, University of Illinois, Urbana, Ill.

[14] Kambayashi, Y., and J.N. Culliney, "NOR network transduction procedures based on connectable and disconnectable conditions (Principles of NOR network transduction programs NETTRA-G1 and NETTRA-G2)," to appear as Report. Dept. of Computer Science, University of Illinois.

[15] Y. Kambayashi, H.C. Lai, J.N. Culliney and S. Muroga, "NOR network transduction by error compensation (Principles of NOR network transduction programs NETTRA-E1, NETTRA-E2, and NETTRA-E3)," to appear as Report, Dept. of Computer Science, Univ. of Illinois, Urbana, Ill.

[16] Kambayashi, Y., H.C. Lai, and S. Muroga, "Transformation of NOR networks," to appear as Report. Department of Computer Science, University of Illinois, Urbana, Ill.,

[17] Kambayashi, Y., and S. Muroga, "Network transduction based on permissible functions (General principles of NOR network transduction NETTRA programs)," UIUCDCS-R-76-804, Dept. of Computer Science, University of Illinois, June 1976.

[18] Lai, H.C., "A study of current logic design problems, part II," Ph.D. thesis, 1976.

[19] Lai, H.C., "Program manual: NOR network transduction by generalized gate merging and substitution (Reference manual of NOR network transduction programs NETTRA-G3 and NETTRA-G4)," UIUCDCS-R-75-714, Dept. of Computer Science, Univ. of Illinois, Urbana, Ill., April 1975.

[20] Lai, H.C. and J.N. Culliney, "Program manual: NOR network pruning procedures using permissible functions (Reference manual of NOR network transduction programs NETTRA-PG1, NETTRA-P1, and NETTRA-P2), UIUCDCS-R-74-686, Dept of Computer Science, Univ. of Illinois, Urbana, Ill., Nov. 1974.

[21] Lai, H.C. and J.N. Culliney, "Program manual: NOR network transduction by error-compensation (Reference manual of NOR network transduction programs NETTRA-E1, NETTRA-E2, and NETTRA-E3)," Report No. UIUCDCS-R-75-732, Dept. of Computer Science, Univ. of Illinois, Urbana, Ill., June 1975.

[22] Lai, H.C. and Y. Kambayashi, "NOR network transduction by generalized gate merging and substitution (Principles of NOR network transduction programs NETTRA-G3 and NETTRA-G4)," UIUCDCS-R-75-728, Dept. of Computer Science, Univ. of Illinois, Urbana, Ill., June 1975.

[23] Lee, H. and E.S. Davidson, "A transform for NAND network design," IEEE Trans., vol. C-21, pp. 12-20, Jan. 1972.

[24] Legge, J.G., "The design of NOR-networks under fan-in and fan-out constraints (A program manual for FIFOTRAN-G1)", Report No. 661, Department of Computer Science, University of Illinois, Urbana, Ill., June 1974.

[25] Liu, T.K., K.R. Hohulin, L. Shiau and S. Muroga, "Optimal one-bit full adders with different types of gates," IEEE Transactions on Computers, Vol. C-23, No. 1, January 1974, pp. 63-70.

[26] McCluskey, E.J. and T.C. Bartee, "A survey of switching circuit theory," McGraw-Hill, New York, 1962.

[27] McCluskey, E.G., "Logical design theory of NOR gate networks with no-complemented inputs," Proc. 4th Ann. Symp. on Switching Circuit Theory and Logical Design, pp. 137-148, 1963.

[28] Maley, G.A. and J. Earle, The Logical Design of Transistor Digital Computers, Englewood Cliffs, N.J.: Prentice Hall, 1963.

[29] Muroga, S., "Logical design of optimal digital networks by integer programming," Chapter 5 in Advances in Information Systems Science, Vol. 3, edited by J.T. Tou, Plenum Press, pp. 283-348, 1970.

[30] Muroga, S., Threshold Logic and Its Applications, Chapter 14, Wiley-Interscience, 1971.

[31] Muroga, S., Logic design and switching theory, class notes, Dept. of Computer Science, Univ. of Illinois, Urbana, Ill.

[32] Muroga, S. and T. Ibaraki, "Logical design of an optimum network by integer linear programming - Part I," Report No. 264, Dept. of Computer Science, Univ. of Illinois, Urbana, Ill., July 1968.

[33] Muroga, S. and T. Ibaraki, "Logical design of an optimum network by integer linear programming - Part II," Report No. 289, Dept. of Computer Science, Univ. of Illinois, Urbana, Ill., Dec. 1968.

[34] Muroga, S. and T. Ibaraki, "Design of optimum switching networks by integer programming," IEEE Trans., Vol. C-21, pp. 573-582, June 1972.

[35] Nakagawa, H.C. Lai and S. Muroga, "Pruning and branching methods for designing optimal networks by the branch-and-bound method," Report No. 471, Dept. of Computer Science, Univ. of Illinois, Urbana, Ill., Aug. 1971. Also International Journal of Computer and Information Sciences, Sept. 1974.

[36] Nakagawa, T. and S. Muroga, "Comparison of the Implicit Enumeration Method and the Branch-and-Bound Method for Logical Design", Report No. UIUCDCS-R-71-455, Department of Computer Science, University of Illinois, Urbana, Ill., June 1971.

[37] Nakagawa, T. and H.C. Lai, "A branch-and-bound algorithm for optimal NOR networks (The algorithm description)," Report No. 438, Dept. of Computer Science, University of Illinois, Urbana, Ill., April 1971.

[38] Plangsiri, B., "NOR network transduction procedures: "Merging of Gates" and "Substitution of Gates" for fan-in and fan-out restricted networks (NETTRA-G3-FIFO and NETTRA-PG1-FIFO)," Report No. UIUCDCS-R-74-688, Dept. of Computer Science, University of Illinois, Urbana, Ill. Dec. 1974.

[39] Su, Y.H. and C. W. Nam "Computer-aided synthesis of multiple output multi-level NAND networks with fan-in and fan-out constraints", IEEE Trans. Comput., Vol. C-20, December 1971.

[40] Tison, P., "Generalization of consensus theory and application to the minimization of Boolean functions," IEEE Tec, August 1967, pp. 446-456.

[41] Torng, H.C., Theory and Logic Design, Addison-Wesley, pp. 112-126, 1972.

[42] White, B.E., "Efficient realization of Boolean functions by pruning NAND trees," September 1969, Lincoln. Lab. Report.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-77-885	2.	3. Recipient's Accession No.

4. Title and Subtitle

NOR NETWORK TRANSDUCTION SYSTEM
(Principles of the NETTRA System)

5. Report Date
August 1977

6.

7. Author(s)
K.C. Hu

8. Performing Organization Rept. No.

9. Performing Organization Name and Address
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No.
NSF DCR73-03421

12. Sponsoring Organization Name and Address
National Science Foundation
1800 G Street, N.W.
Washington, D.C. 20550

13. Type of Report & Period Covered
Technical

14.

15. Supplementary Notes

16. Abstracts

The network transduction programs, including NETTRA-PG1, -P1, -P2, -G1, -G2, -G3, -G4, -E1, and -E2 are combined as a large program system named the NETTRA system (NETwork TRAnsduction system). The NETTRA system can design near-optimal, multiple-output, multi-level and loop-free NOR(NAND) networks under fan-in/fan-out restrictions and/or level restriction. Given function(s) may be completely or incompletely specified and both complemented and uncomplemented external variables are permitted as inputs. The user can specify the control sequence (the types of the initial network methods and the types and the order of the transduction procedures to be applied) to solve his problem. Besides, four control sequences are provided for the users who are not interested in the details of how to specify the control sequence. Facilities for treating unfinished jobs due to the expiration of computation time are also provided by the system.

17. Key Words and Document Analysis. 17a. Descriptors

Logic design, logic circuits, logic elements, programs (computer)

17b. Identifiers/Open-Ended Terms

Computer-aided design, transduction procedures, transformations, near-optimal networks, optimal networks, permissible functions, CSPF, fan-in, fan-out, level, NOR, NAND, NETTRA system, integer programming logic design

17c. COSATI Field/Group

18. Availability Statement Release Unlimited	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 217
	20. Security Class (This Page UNCLASSIFIED	22. Price